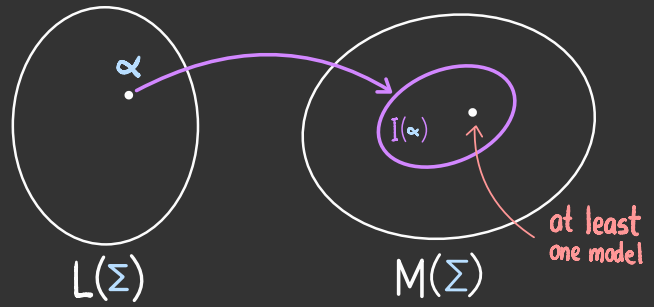
search-based satisfiability

CSCI 373

satisfiability:

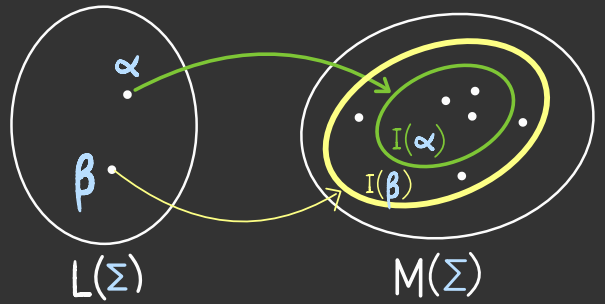
a sentence $\alpha \in L(\Sigma)$ is satisfiable if its interpretation has at least one model, i.e. $I(\alpha) \neq \{\}$



entailment:

for sentences $\alpha, \beta \in L(\Sigma)$,

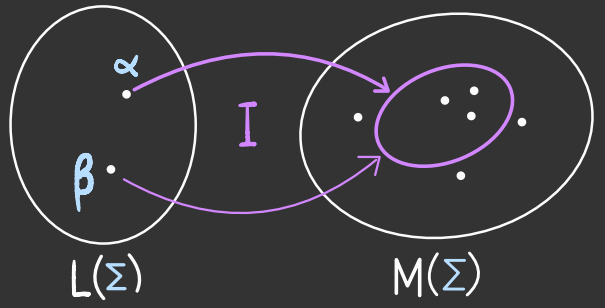
$\alpha \models \beta$ if and only if $I(\alpha) \subseteq I(\beta)$



equivalence:

for sentences $\alpha, \beta \in L(\Sigma)$,

$\alpha \equiv \beta$ if and only if $I(\alpha) = I(\beta)$

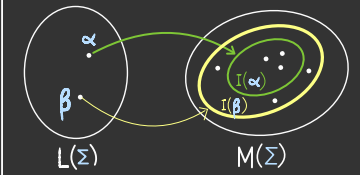


$\alpha \models \beta$

if and only if

$\alpha \wedge \neg\beta$ is unsatisfiable

entailment

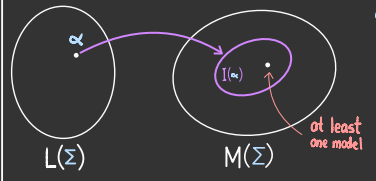


$\alpha \models \beta$

if and only if

$I(\alpha) \subseteq I(\beta)$

satisfiability



α is satisfiable

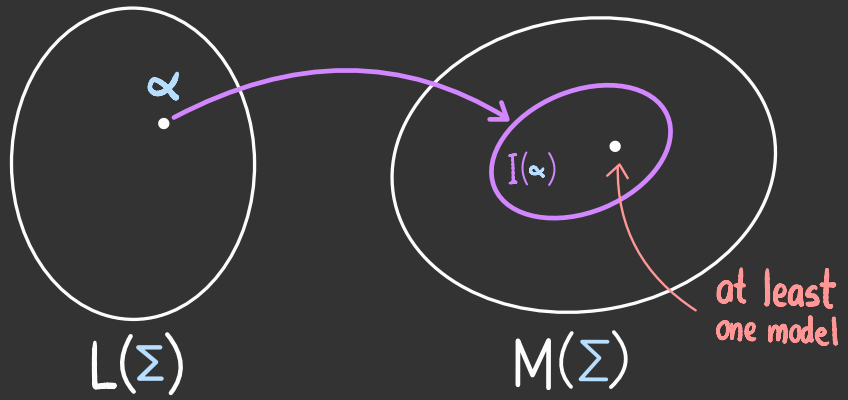
if and only if

$I(\alpha) \neq \{\}$

α is unsatisfiable

if and only if

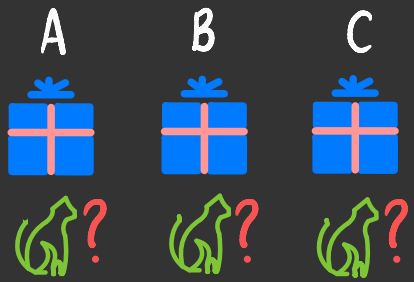
$\alpha \models \bot$



satisfiability

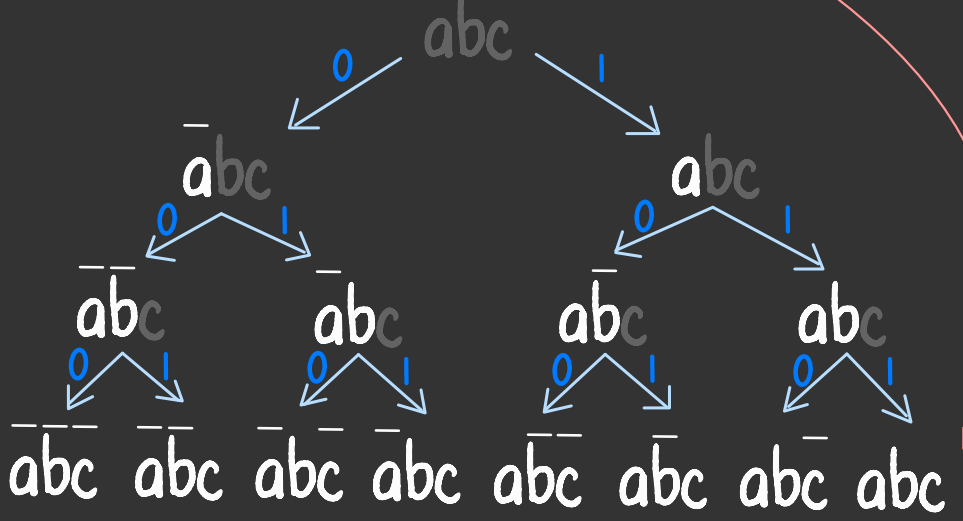
a "satisfiability solver" is an algorithm that computes whether sentence α has a model in its interpretation

how might we create a "sat solver" using search?

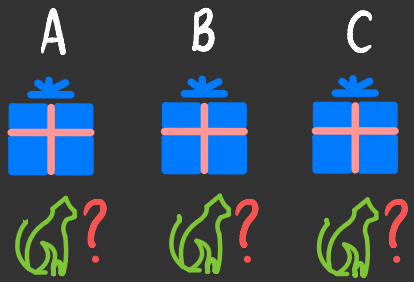


"the cat is in exactly one box"

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



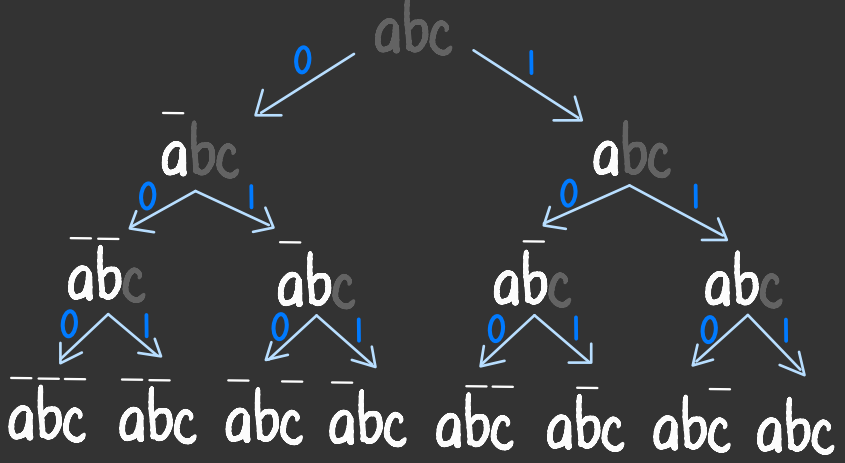
at the leaves, we check if the model satisfies the sentence

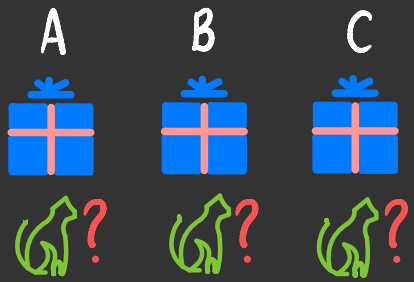


"the cat is in exactly one box"

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

what is an appropriate **search** algorithm?

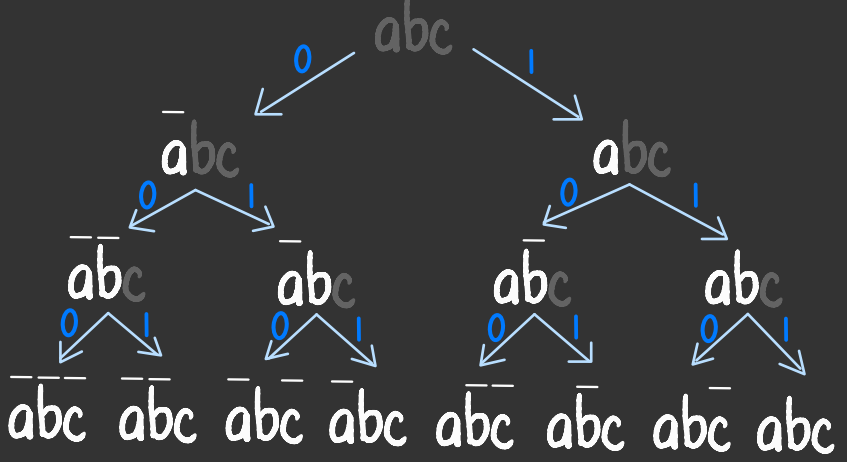




"the cat is in exactly one box"

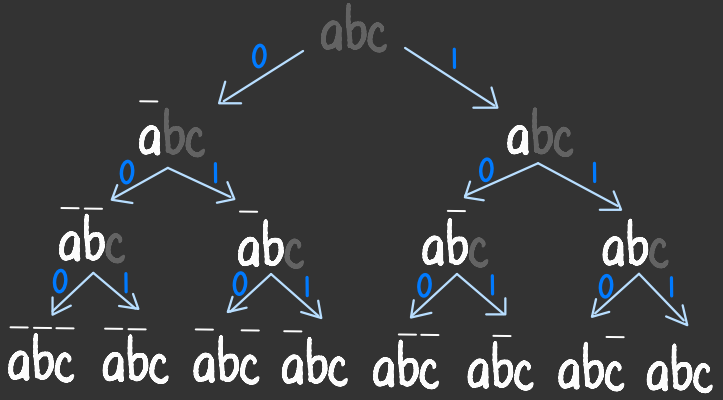
$$(A \vee B \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg C)$$

what is an appropriate **search** algorithm?



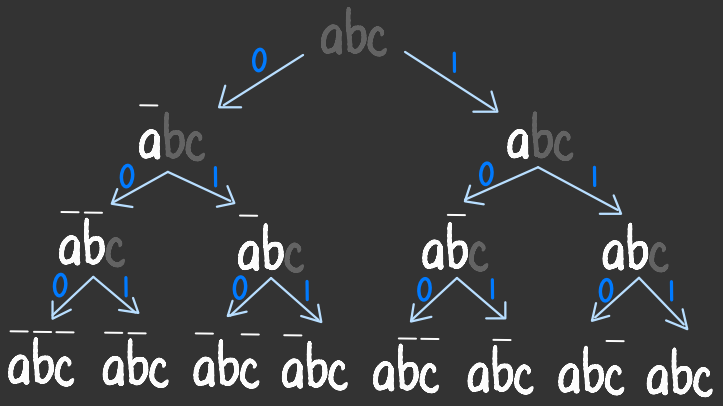
acyclic and known solution depth, so

dfs



dfs $(M = (Q, \Sigma, \Delta, q_0, F))$:

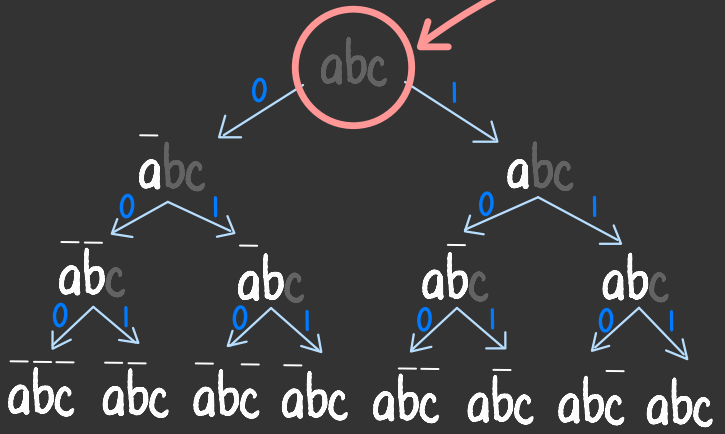
- container = new **stack**()
- container.put(q_0)
- repeat:
  - if container.empty() then return **None**
  - q = container.get()
  - if $q \in F$ then return q
  - let $\text{successors}_M(n) = \{ q' \mid (q, \sigma, q') \in \Delta \}$
  - for $q' \in \text{successors}_M(q)$:
    container.put(n')



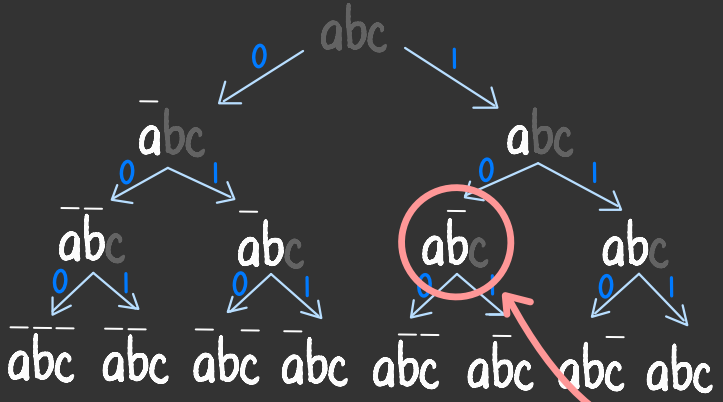
findSatisfyingModel(α):

- container = new stack()
- container.put({})
- repeat:
  - if container.empty() then return "unsatisfiable"
  - m = container.get()
  - if m $\in I(\alpha)$ then return m
  - for $m' \in$ successors(m):
    container.put(m')

initial state:
haven't assigned any signature variables

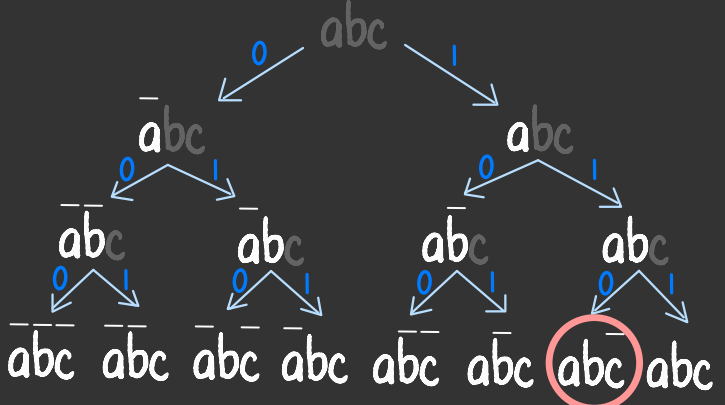



```
findSatisfyingModel(α):
  container = new stack()
  container.put({})
  repeat:
    if container.empty() then return "unsatisfiable"
    m = container.get()
    if m ∈ I(α) then return m
    for m′ ∈ successors(m):
      container.put(m′)
```



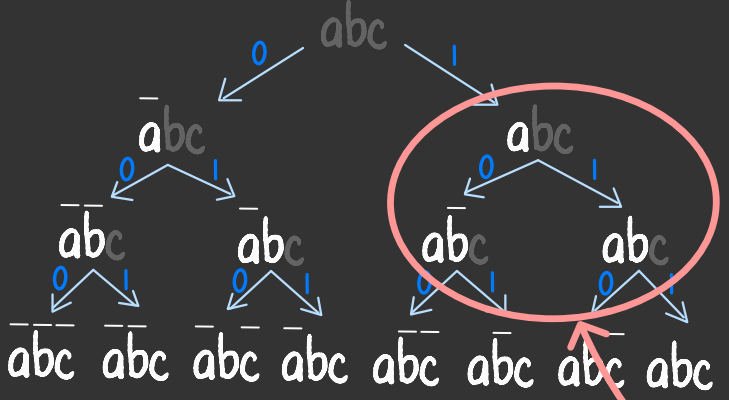
other states:
(partially specified) models

```
findSatisfyingModel(α):
  container = new stack()
  container.put({})
  repeat:
    if container.empty() then return "unsatisfiable"
    m = container.get()
    if m ∈ I(α) then return m
    for m′ ∈ successors(m):
      container.put(m′)
```



goal states:
satisfying models

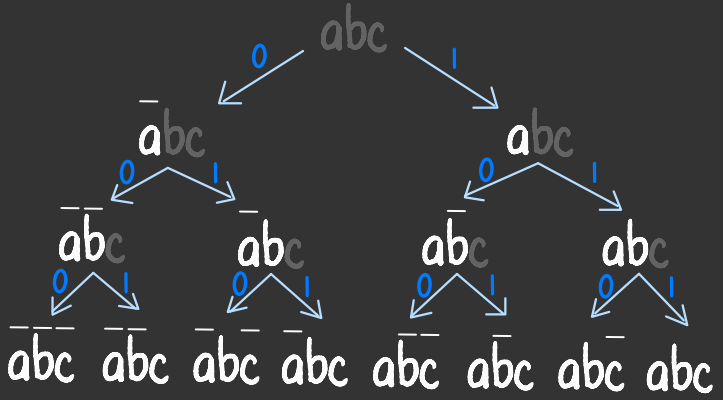
```
findSatisfyingModel(α):
  container = new stack()
  container.put({})
  repeat:
    if container.empty() then return "unsatisfiable"
    m = container.get()
    if m ∈ I(α) then return m
    for m′ ∈ successors(m):
      container.put(m′)
```



findSatisfyingModel(α):

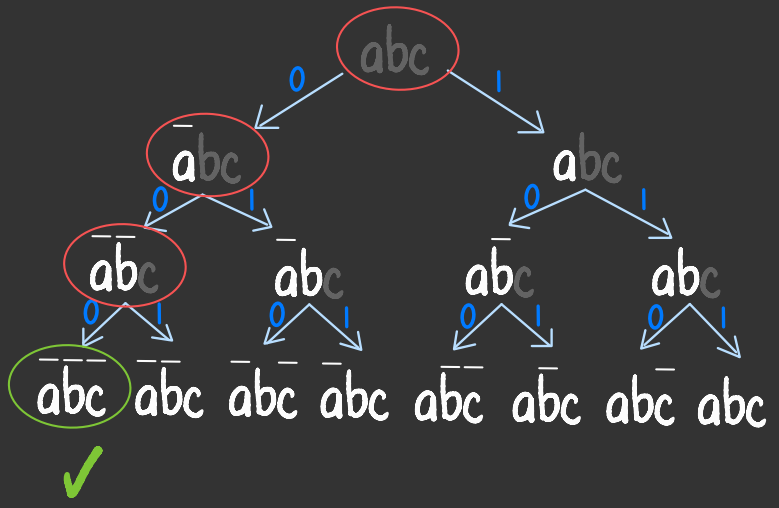
- container = new stack()
- container.put({})
- repeat:
  - if container.empty() then return "unsatisfiable"
  - m = container.get()
  - if $m \in I(\alpha)$ then return m
  - for $m' \in$ successors(m):
    container.put(m')

successors:
assign another
signature variable



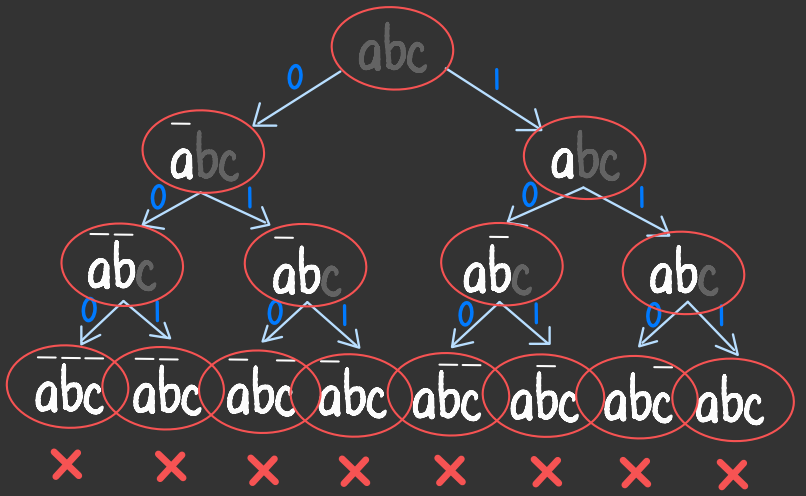
is this approach faster for satisfiable or unsatisfiable sentences?

if α is satisfiable,
the search-based approach
might terminate early

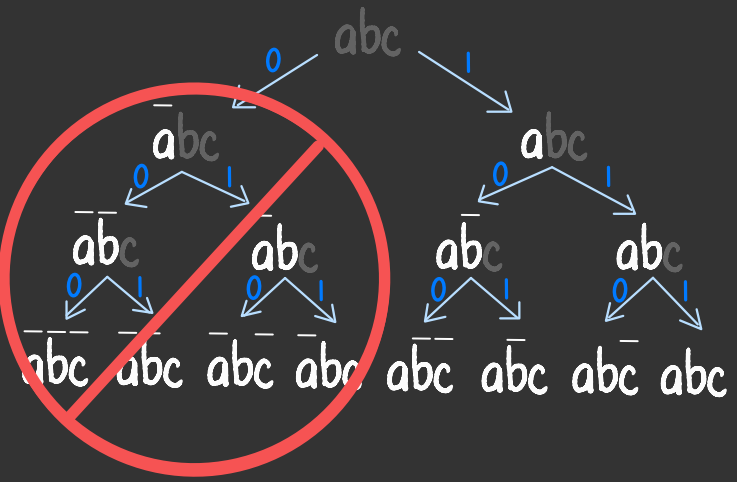


✓

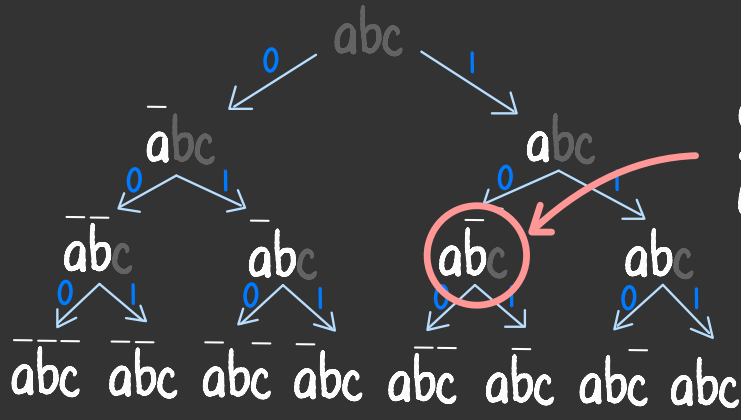
if α is unsatisfiable,
the search-based approach
visits every node



× × × × × × × ×

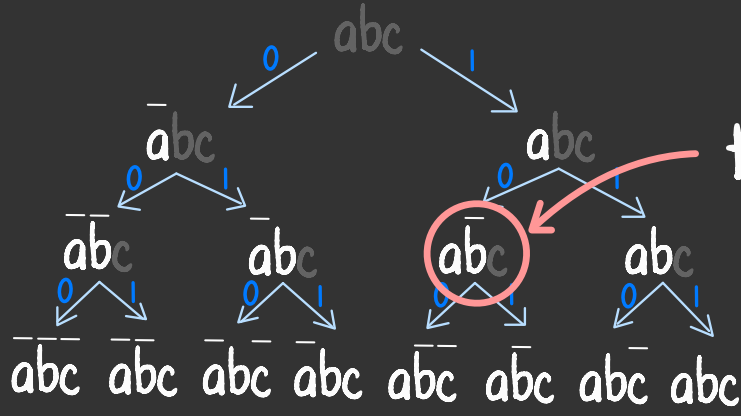


but perhaps we can find ways to prune **large subtrees** of the search tree



$\{A \mapsto 1, B \mapsto 0\}$

define a partial model of signature Σ
as a function $m : \Sigma' \to \{0, 1\}$ where $\Sigma' \subseteq \Sigma$



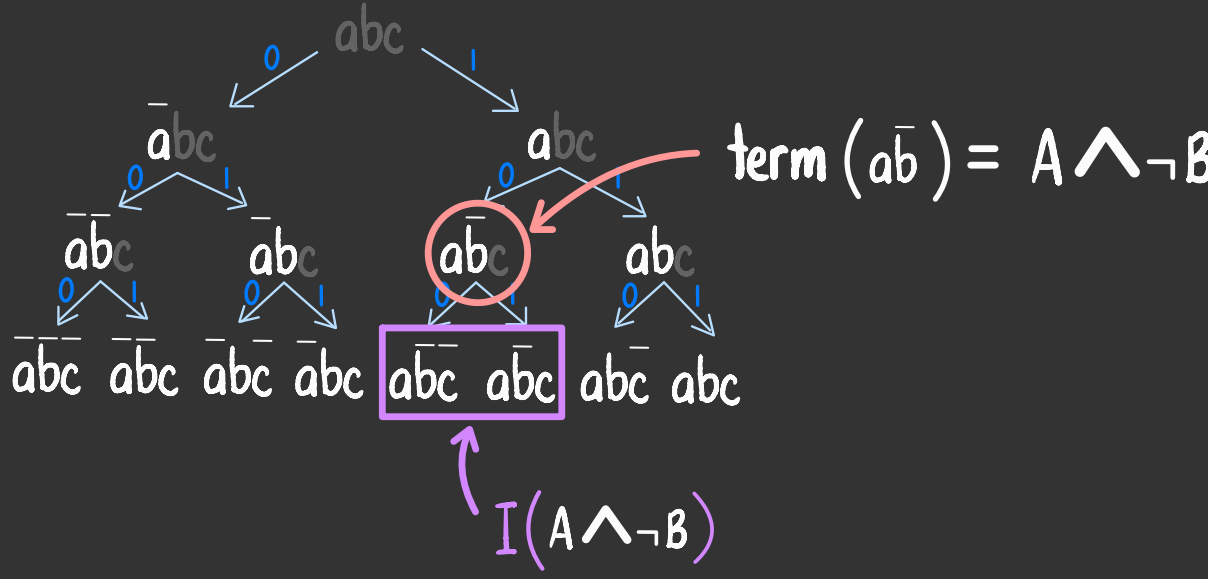
$\text{term}(a\bar{b}) = A \wedge \neg B$

for partial model $m: \Sigma' \to \{0,1\}$, define:

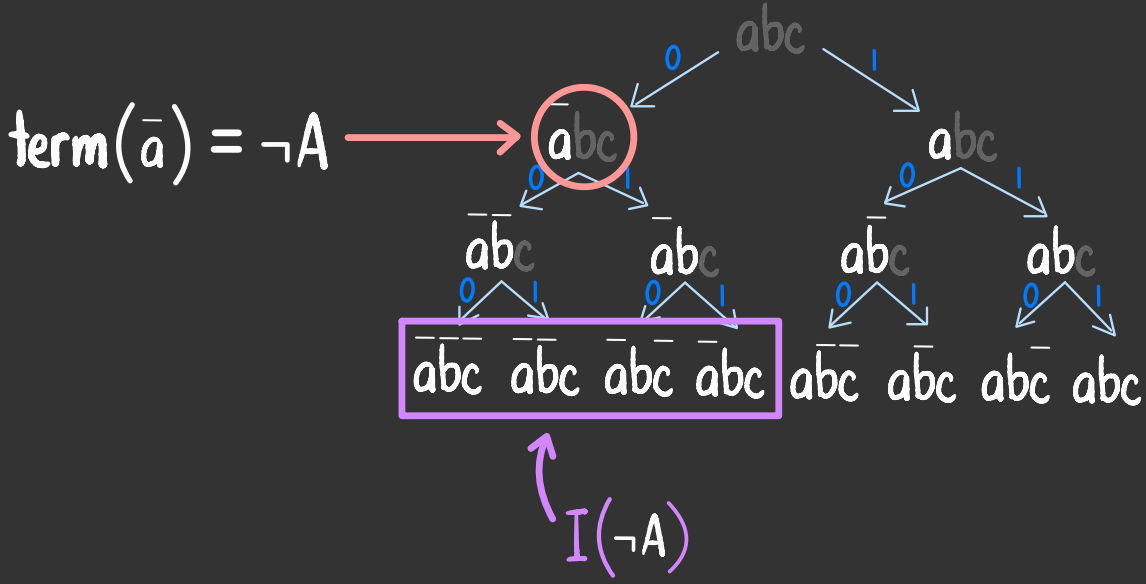
$$\text{literal}(m,\sigma) = \begin{cases} \sigma & \text{if } m(\sigma) = 1 \\ \neg\sigma & \text{if } m(\sigma) = 0 \end{cases}$$

where $\sigma \in \Sigma'$

$$\text{term}(m) = \bigwedge_{\sigma \in \Sigma'} \text{literal}(m,\sigma)$$

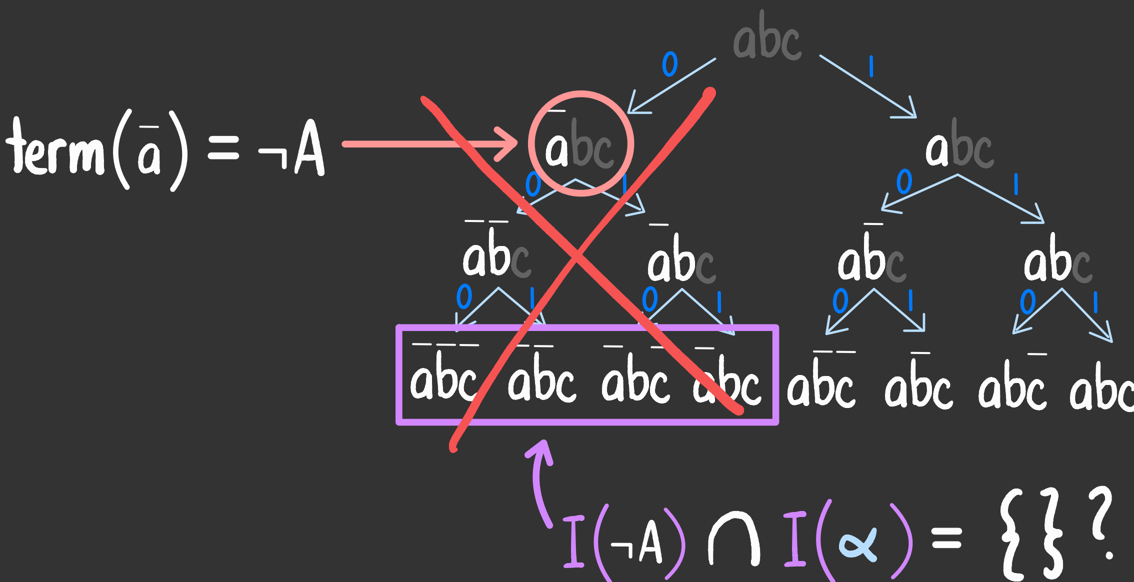


the leaves of search node m
are the models of I(term(m))



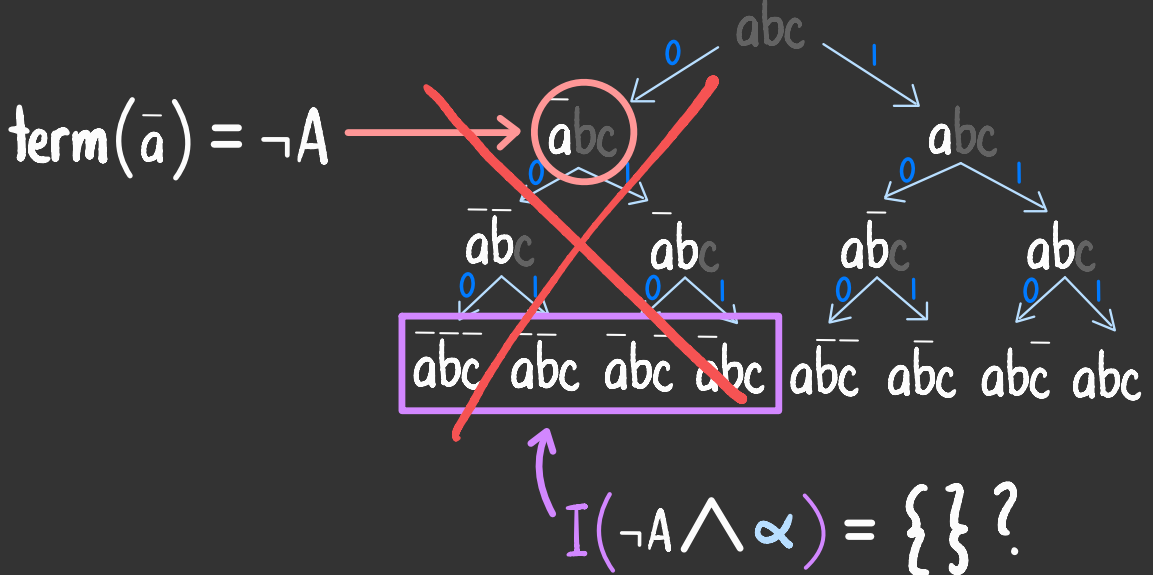
the leaves of search node m
are the models of I(term(m))

findSatisfyingModel(α)



if we can establish that none of the models of $I(term(m))$ are models in $I(\alpha)$, then we can prune the subtree

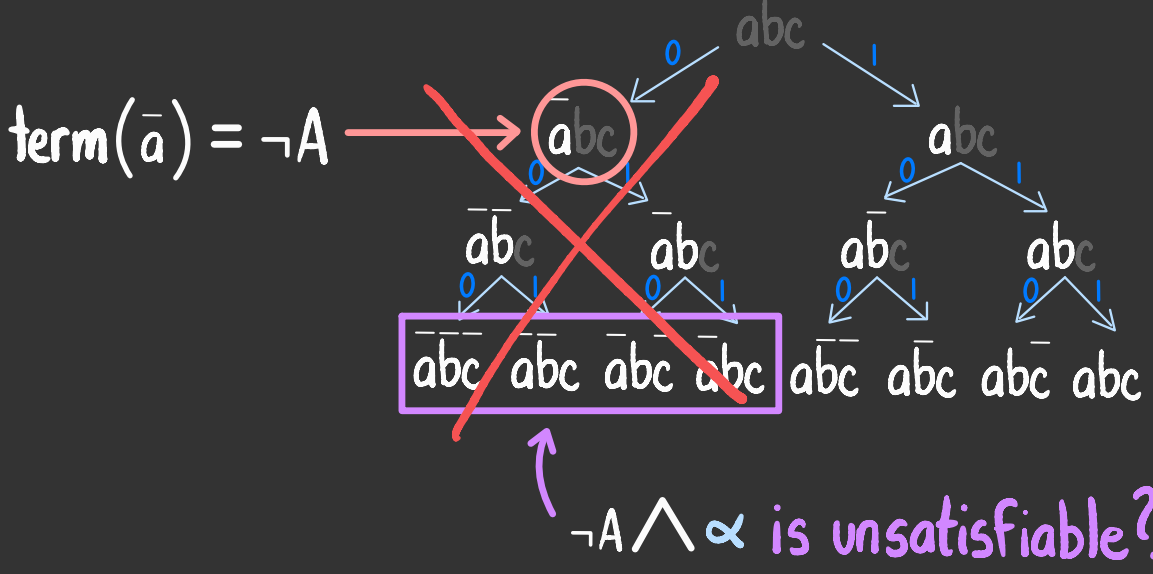
find Satisfying Model (α)



if we can establish that $I(\text{term}(m) \wedge \alpha) = \{\}$,

then we can prune the subtree

find Satisfying Model (α)



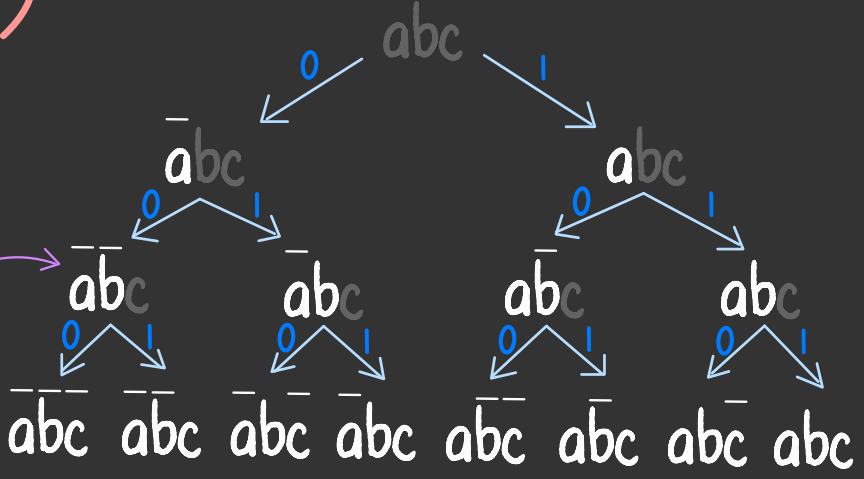
if we can establish that term(m) $\wedge \alpha$ is unsatisfiable,

then we can prune the subtree

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

cnf

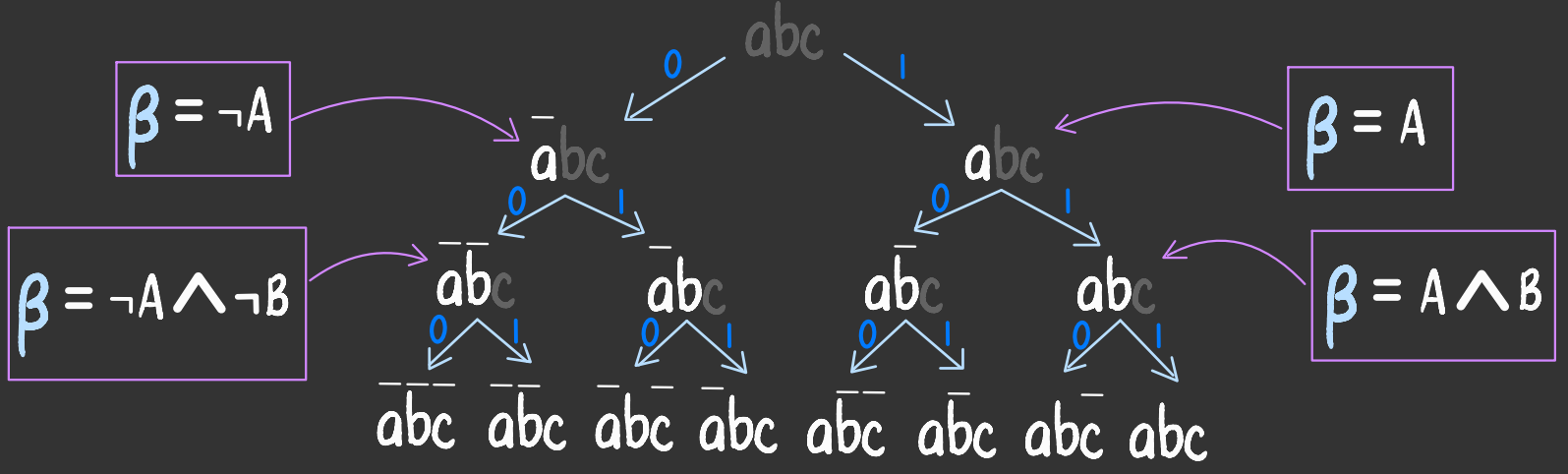
$$\beta = \neg A \wedge \neg B$$



also cnf

$$\beta \wedge \alpha = \neg A \wedge \neg B \wedge (A \vee B \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg C)$$

find Satisfying Model (α)



we can prune search nodes if

we can determine that a cnf sentence

$IM(\beta \wedge \alpha)$ is unsatisfiable

Can we find a quicker way to identify unsatisfiability?

findSatisfyingModel(α):

- container = new stack()
- container.put({})
- repeat:
  - if container.empty() then return "unsatisfiable"
  - m = container.get()
  - if m $\in$ I(α) then return m
  - for m′ $\in$ successors(m):
    container.put(m′)

can we find a **quicker** way to identify unsatisfiability?

```
findSatisfyingModel(α):
  container = new stack()
  container.put({})
  repeat:
    if container.empty() then return "unsatisfiable"
    m = container.get()
    if m ∈ I(α) then return m
    if α ∧ term(m) is unsatisfiable:
      do nothing
    else:
      for m' ∈ successors(m):
        container.put(m')
```

this quick unsatisfiability test needs to be sound, i.e. if it says a sentence is unsatisfiable, then it must be unsatisfiable

```
findSatisfyingModel(α):
  container = new stack()
  container.put({})
  repeat:
    if container.empty() then return "unsatisfiable"
    m = container.get()
    if m ∈ I(α) then return m
    if α ∧ term(m) is unsatisfiable:
      do nothing
    else:
      for m′ ∈ successors(m):
        container.put(m′)
```

but it does not have to be complete,

i.e., if it says a sentence is satisfiable, then it doesn't have to be

it just means we don't prune a node that we could have pruned

findSatisfyingModel(α):

- container = new stack()
- container.put({})
- repeat:
  - if container.empty() then return "unsatisfiable"
  - m = container.get()
  - if $m \in I(\alpha)$ then return m
  - if $\alpha \wedge$ term(m) is unsatisfiable:
    - do nothing
  - else:
    - for $m' \in$ successors(m):
      - container.put(m')

Can we find a faster way to determine whether a sentence is unsatisfiable?