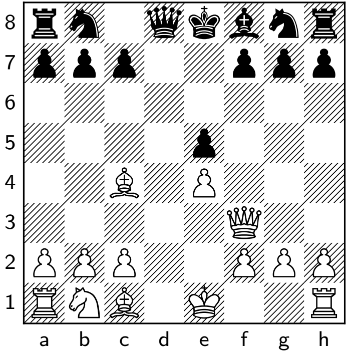
Concrete Interpretation of Chess

Möbius Strip Reverse Engineering

February 26, 2018

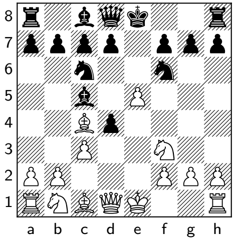
A Static Analysis of Chess



- We analyze the game of chess exactly how we analyze computer programs: by **abstract interpretation**.
- Chess allows us to introduce most of the concepts in a simpler and highly visual context.

State Space

Description of a Moment in a Game



- Any point during a game is characterized by:
  1. Whose turn it is;
     - Let's call this **pc**, for **player color**.
     - Define a set $PC = \{White, Black\}$
     - Now $\mathbf{pc} \in PC$.
  2. Where each piece is, or whether it has been captured.
     - This is the **board configuration**, written σ.
- Hence $\langle \mathbf{pc}, \sigma \rangle$ describes a moment in the game.

Representing State Spaces

How to Describe Board Configurations

- Define a variable for every piece. $Vars =$

$$\left\{\begin{array}{llll} Pawn_{W,1} & Pawn_{W,2} & Pawn_{W,3} & Pawn_{W,4} \\ Pawn_{W,5} & Pawn_{W,6} & Pawn_{W,7} & Pawn_{W,8} \\ Rook_{W,1} & Rook_{W,2} & Knight_{W,1} & Knight_{W,2} \\ Bishop_{W,1} & Bishop_{W,2} & Queen_{W} & King_{W} \\ Pawn_{B,1} & Pawn_{B,2} & Pawn_{B,3} & Pawn_{B,4} \\ Pawn_{B,5} & Pawn_{B,6} & Pawn_{B,7} & Pawn_{B,8} \\ Rook_{B,1} & Rook_{B,2} & Knight_{B,1} & Knight_{B,2} \\ Bishop_{B,1} & Bishop_{B,2} & Queen_{B} & King_{B} \end{array}\right\}$$

- Define the set of squares, plus a special *captured* square.
  - $Squares = \{a1, \ldots, a8, \ldots, h1, \ldots, h8, Captured\}$
- A state (**board configuration**) is a function $State : Vars \rightarrow Squares$.

Representing State Spaces

Storing Chess Boards on a Computer

Square	Bits	Square	Bits	Square	Bits	Square	Bits
$a1$	0000000	$a2$	0000001	$a3$	0000010	$a4$	0000011
$a5$	0000100	$a6$	0000101	$a7$	0000110	$a8$	0000111
$b1$	0001000	$b2$	0001001	$b3$	0001010	$b4$	0001011
$b5$	0001100	$b6$	0001101	$b7$	0001110	$b8$	0001111
$c1$	0010000	$c2$	0010001	$c3$	0010010	$c4$	0010011
$c5$	0010100	$c6$	0010101	$c7$	0010110	$c8$	0010111
$d1$	0011000	$d2$	0011001	$d3$	0011010	$d4$	0011011
$d5$	0011100	$d6$	0011101	$d7$	0011110	$d8$	0011111
$e1$	0100000	$e2$	0100001	$e3$	0100010	$e4$	0100011
$e5$	0100100	$e6$	0100101	$e7$	0100110	$e8$	0100111
$f1$	0101000	$f2$	0101001	$f3$	0101010	$f4$	0101011
$f5$	0101100	$f6$	0101101	$f7$	0101110	$f8$	0101111
$g1$	0110000	$g2$	0110001	$g3$	0110010	$g4$	0110011
$g5$	0110100	$g6$	0110101	$g7$	0110110	$g8$	0110111
$h1$	0111000	$h2$	0111001	$h3$	0111010	$h4$	0111011
$h5$	0111100	$h6$	0111101	$h7$	0111110	$h8$	0111111
Captured	1000000						

- There are 65 locations where a piece might be located. This can be represented by 7 bits.
  - 64 squares, plus the *Captured* location.
- There are 32 pieces.
- Hence, $7 * 32 = 224$ bits per board.
  - No claim of optimality is being made.

Initial States

Initial Board Configurations

- The game starts from the **initial configuration**. (Other systems may have more than one initial configuration).

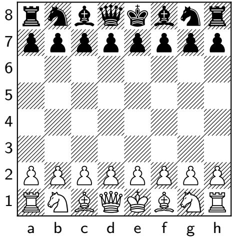
$\Big\langle$ White,  $\Big\rangle$

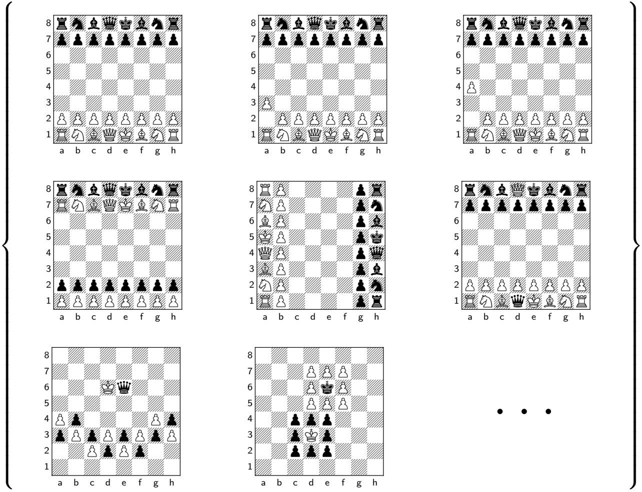
Table: The initial configuration, σ_{init}

Piece	Square	Piece	Square	Piece	Square	Piece	Square
$Pawn_{W,1}$	$a2$	$Pawn_{W,2}$	$b2$	$Pawn_{W,3}$	$c2$	$Pawn_{W,4}$	$d2$
$Pawn_{W,5}$	$e2$	$Pawn_{W,6}$	$f2$	$Pawn_{W,7}$	$g2$	$Pawn_{W,8}$	$h2$
$Rook_{W,1}$	$a1$	$Rook_{W,2}$	$h1$	$Knight_{W,1}$	$b1$	$Knight_{W,2}$	$g1$
$Bishop_{W,1}$	$c1$	$Bishop_{W,2}$	$f1$	$Queen_{W}$	$d1$	$King_{W}$	$e1$
$Pawn_{B,1}$	$a7$	$Pawn_{B,2}$	$b7$	$Pawn_{B,3}$	$c7$	$Pawn_{B,4}$	$d7$
$Pawn_{B,5}$	$e7$	$Pawn_{B,6}$	$f7$	$Pawn_{B,7}$	$g7$	$Pawn_{B,8}$	$h7$
$Rook_{B,1}$	$a8$	$Rook_{B,2}$	$h8$	$Knight_{B,1}$	$b8$	$Knight_{B,2}$	$g8$
$Bishop_{B,1}$	$c8$	$Bishop_{B,2}$	$f8$	$Queen_{B}$	$d8$	$King_{B}$	$e8$

The Alphabet

All Possible Board Configurations

The **alphabet** Σ is the set of all possible states.



State Transitions

Describing Changes to the Board Configuration

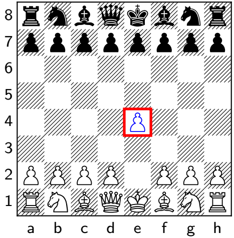


Figure: $\sigma_{init}[Pawn_{W,5} \mapsto e4]$

- Moves are characterized as **state updates**.
- Given some existing state σ, we write $\sigma[P_1 \mapsto \ell_1, \ldots, P_n \mapsto \ell_n]$ for the state that is the same as σ, except each of the pieces P_i has moved to locations ℓ_i.

Semantics of State Transitions

Describing Legal Piece Moves

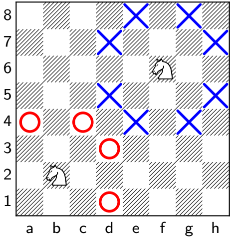


Figure: Legal knight moves

- The laws of chess dictate which moves are valid.
  - E.g., a knight can only move in an "L-shape", and only if such a move would stay within the boundaries of the board.
- We will specify the rules as **state transitions**.

Inference Rules

Individual Chess Moves

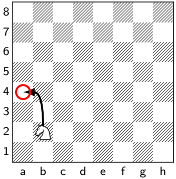


Figure: This move is formalized below as an **inference rule**.

k abbreviates $Knight_{W,1}$.

$$\frac{\sigma(k) \neq Captured}{\quad} \; Knight_{W,1} - NWN - Move$$

- If the **premises** above the bar are true:
  1. $Knight_{W,1}$ has not been captured

Inference Rules

Individual Chess Moves

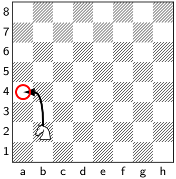


Figure: This move is formalized below as an **inference rule**.

k abbreviates $Knight_{W,1}$.

$$\frac{\sigma(k) \neq Captured \quad file(\sigma(k)) \geq b}{} \; Knight_{W,1} - NWN - Move$$

- If the **premises** above the bar are true:
  1. $Knight_{W,1}$ has not been captured
  2. $Knight_{W,1}$ is in the b^{th} file or above

Inference Rules

Individual Chess Moves

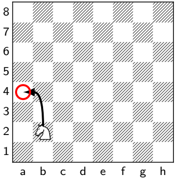


Figure: This move is formalized below as an **inference rule**.

k abbreviates $Knight_{W,1}$.

$$\frac{\sigma(k) \neq Captured \quad file(\sigma(k)) \geq b \quad rank(\sigma(k)) \leq 6}{} \; Knight_{W,1} - NWN - Move$$

- If the **premises** above the bar are true:
  1. $Knight_{W,1}$ has not been captured
  2. $Knight_{W,1}$ is in the b^{th} file or above
  3. $Knight_{W,1}$ is in the 6^{th} rank or below

Inference Rules

Individual Chess Moves

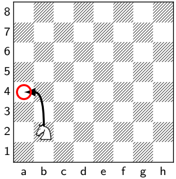


Figure: This move is formalized below as an **inference rule**.

k abbreviates $Knight_{W,1}$.

$$\frac{\sigma(k) \neq Captured \quad file(\sigma(k)) \geq b \quad rank(\sigma(k)) \leq 6 \quad \neg occupied(\sigma(k) + 2_N + 1_W)}{} \; Knight_{W,1} - NWN - Move$$

- If the **premises** above the bar are true:
  1. $Knight_{W,1}$ has not been captured
  2. $Knight_{W,1}$ is in the b^{th} file or above
  3. $Knight_{W,1}$ is in the 6^{th} rank or below
  4. No piece is located at the red circle relative to $Knight_{W,1}$

Inference Rules

Individual Chess Moves

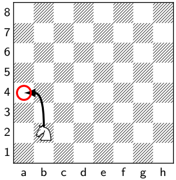


Figure: This move is formalized below as an **inference rule**.

k abbreviates $Knight_{W,1}$.

$$\frac{\sigma(k) \neq Captured \quad file(\sigma(k)) \geq b \quad rank(\sigma(k)) \leq 6 \quad \neg occupied(\sigma(k) + 2_N + 1_W)}{\sigma[k \mapsto \sigma(k) + 2_N + 1_W]} \; Knight_{W,1} - NWN - Move$$

- If the **premises** above the bar are true:
    1. $Knight_{W,1}$ has not been captured
    2. $Knight_{W,1}$ is in the b^{th} file or above
    3. $Knight_{W,1}$ is in the 6^{th} rank or below
    4. No piece is located at the red circle relative to $Knight_{W,1}$
- Then the **conclusion** below the bar is true:
    - Moving to the location described in 4 is valid.

Operational Semantics

All Chess Moves

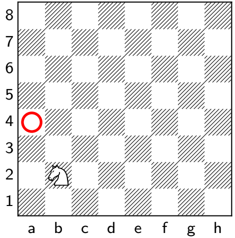
$\rightarrow Pawn_{W,1}{-}N$	$\rightarrow Pawn_{W,1}{-}NN$	$\rightarrow Pawn_{W,2}{-}N$	$\rightarrow Pawn_{W,2}{-}NN$
$\rightarrow Knight_{W,1}{-}NWN$	$\rightarrow Knight_{W,1}{-}WNW$	$\rightarrow Queen_{W}{-}N1$	$\rightarrow Queen_{W}{-}N2$
$\rightarrow Queen_{W}{-}N3$	$\rightarrow Queen_{W}{-}W1$	$\rightarrow Queen_{W}{-}E1$	$\rightarrow Queen_{W}{-}S1$
$\rightarrow Queen_{W}{-}NW1$	$\rightarrow Queen_{W}{-}NE1$	$\rightarrow Queen_{W}{-}SE1$	$\rightarrow Queen_{W}{-}SW1$
$\rightarrow King_{W}{-}N$	$\rightarrow King_{W}{-}W$	$\rightarrow King_{W}{-}E$	$\cdots$

Figure: Partial listing of all legal chess moves

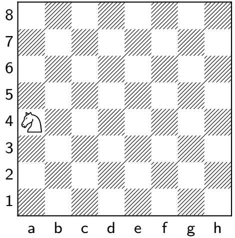
- Write $b_1 \rightarrow_{(Move)} b_2$ if applying $(Move)$ to board b_1 yields board b_2.
- The collection of all legal chess moves as inference rules is called the **operational semantics** of chess.

The Transition Relation, $\rightarrow_{Chess}$

Legal Chess Moves



(a) σ_1



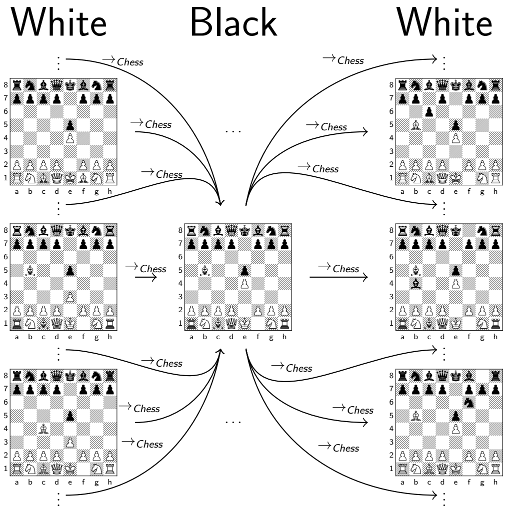
(b) σ_2

Figure: A legal move: $\sigma_1 \rightarrow_{Chess} \sigma_2$

- If one configuration (σ_2) can be obtained from another (σ_1) by applying one valid state transition, we write:
  1. $\sigma_1 \rightarrow_{Chess} \sigma_2$, OR
  2. $\tau(\sigma_1, \sigma_2)$, or $\sigma_1 \ \tau \ \sigma_2$, or $(\sigma_1, \sigma_2) \in \tau$.

State Transition Diagram

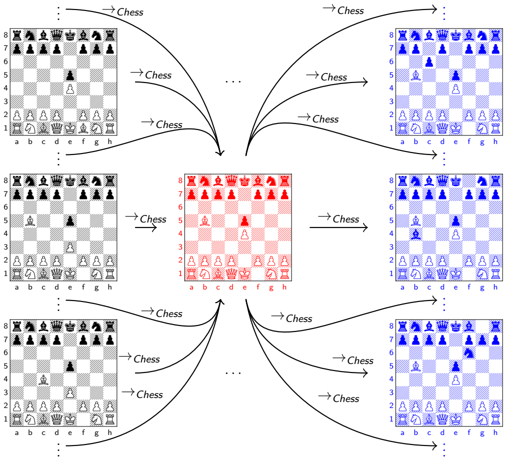
Graph of Legal Chess Moves



- Depicts every valid state transition.

Successor States $post_{\rightarrow_{Chess}}$, Visualized

All Legal Chess Moves from a Given Position



The successor relationship $post_{\rightarrow_{Chess}}$ yields the set of all states that result from a given state under one application of the transition relation (i.e. $\rightarrow_{Chess}$).

Successor States $post_{\rightarrow_{Chess}}$

All Legal Chess Moves from a Given Position

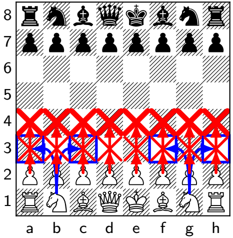


Figure: All possible opening moves

$post_{\rightarrow_{Chess}}(White, \sigma_{init})$, all legal moves from $\langle White, \sigma_{init} \rangle$:

$$\left\{ \begin{array}{llll} \sigma_{init}[Pawn_{W,1} \mapsto a3] & \sigma_{init}[Pawn_{W,1} \mapsto a4] & \sigma_{init}[Pawn_{W,2} \mapsto b3] & \sigma_{init}[Pawn_{W,2} \mapsto b4] \\ \sigma_{init}[Pawn_{W,3} \mapsto c3] & \sigma_{init}[Pawn_{W,3} \mapsto c4] & \sigma_{init}[Pawn_{W,4} \mapsto d3] & \sigma_{init}[Pawn_{W,4} \mapsto d4] \\ \sigma_{init}[Pawn_{W,5} \mapsto e3] & \sigma_{init}[Pawn_{W,5} \mapsto e4] & \sigma_{init}[Pawn_{W,6} \mapsto f3] & \sigma_{init}[Pawn_{W,6} \mapsto f4] \\ \sigma_{init}[Pawn_{W,7} \mapsto g3] & \sigma_{init}[Pawn_{W,7} \mapsto g4] & \sigma_{init}[Pawn_{W,8} \mapsto h3] & \sigma_{init}[Pawn_{W,8} \mapsto h4] \\ \sigma_{init}[Knight_{W,1} \mapsto a3] & \sigma_{init}[Knight_{W,1} \mapsto c3] & \sigma_{init}[Knight_{W,2} \mapsto f3] & \sigma_{init}[Knight_{W,2} \mapsto h3] \end{array} \right\}$$

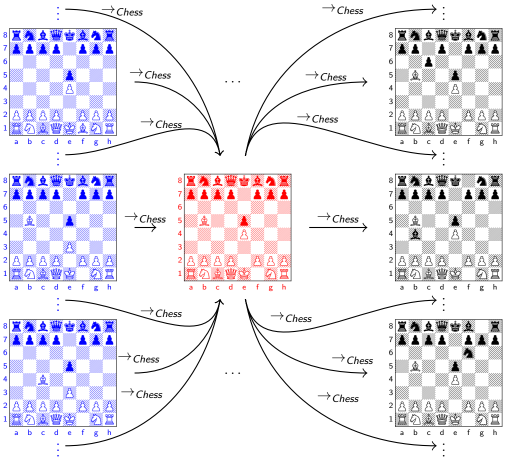
Successor States $post_{\rightarrow_{Chess}}$

All Legal Chess Moves from a Given Position

- In any non-final state, at least one transition is possible; in chess, often more than one.
- Let $post_{\rightarrow_{Chess}}(pc, \sigma) : PC \times State \rightarrow \wp(PC \times State)$ denote the set of all possible legal states after making one transition from $\langle pc, \sigma \rangle$.
- Formally,
  $post_{\rightarrow_{Chess}}(pc, \sigma) = \{\langle pc', \sigma' \rangle \mid \langle pc, \sigma \rangle \rightarrow_{Chess} \langle pc', \sigma' \rangle\}$
- This is the **set of successors** of $\langle pc, \sigma \rangle$ under $\rightarrow_{Chess}$.

Predecessor States $pre_{\rightarrow_{Chess}}$, Visualized

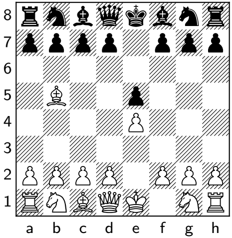
All Legal Chess Moves Leading to a Given Position



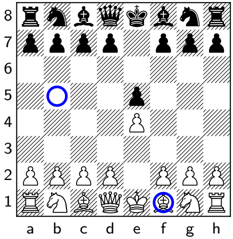
The predecessor relationship $pre_{\rightarrow_{Chess}}$ yields the set of all states that lead to a given state in one application of the transition relation (i.e. $\rightarrow_{Chess}$).

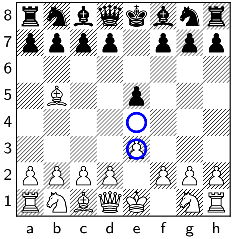
Predecessor States $pre_{\rightarrow Chess}$

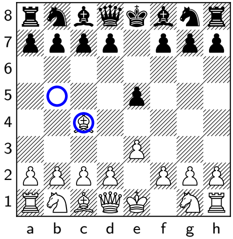
All Legal Chess Moves Leading to a Given Position



The boards below can transition to the one above.







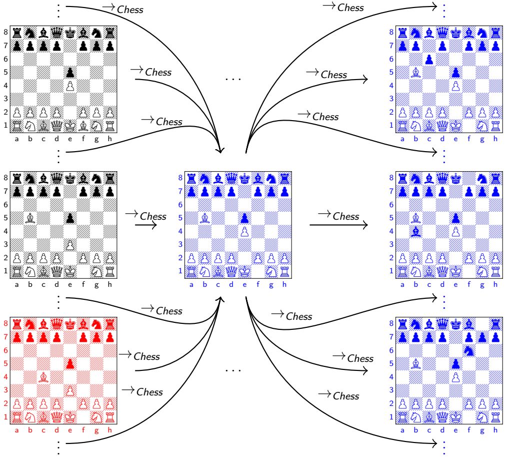
Predecessor States $pre_{\rightarrow_{Chess}}$

All Legal Chess Moves Leading to a Given Position

- Let $pre_{\rightarrow_{Chess}}(pc, \sigma) : PC \times State \rightarrow \wp(PC \times State)$ denote the set of all possible legal states that lead to $\langle pc, \sigma \rangle$ after making one transition.
- Formally,
  $pre_{\rightarrow_{Chess}}(pc, \sigma) = \{\langle pc', \sigma' \rangle \mid \langle pc', \sigma' \rangle \rightarrow_{Chess} \langle pc, \sigma \rangle\}$
- This is the **set of predecessors** of $\langle pc, \sigma \rangle$ under $\rightarrow_{Chess}$.

The Transitive Closure of $\to_{Chess}$, $\to^*_{Chess}$

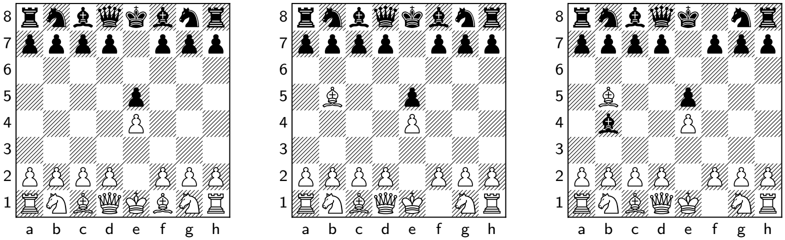
All Positions Reachable from a Given Position



- The **transitive closure** of $\to_{Chess}$, denoted $\to^*_{Chess}$, is the set of all configurations reachable from some configuration via one or more applications of $\to_{Chess}$.

Paths

Sequences of Valid Moves



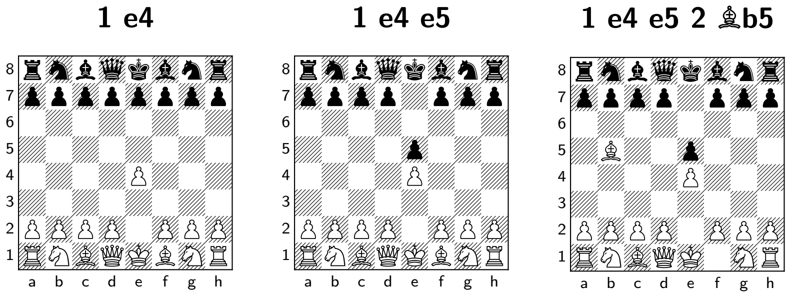
(a) $\langle White, \sigma_2 \rangle$ (b) $\langle Black, \sigma_3 \rangle$ (c) $\langle White, \sigma_4 \rangle$

Figure: A path of length 3.

- A **path** π of length i is a sequence of pairs $\langle pc_1, \sigma_1 \rangle, \langle pc_2, \sigma_2 \rangle, \langle pc_3, \sigma_3 \rangle, \ldots, \langle pc_i, \sigma_i \rangle$ such that:
    1. $\sigma_i \rightarrow_{Chess} \sigma_{i+1}$.
    2. The pc_is alternate colors.
- Write $|\pi|$ for the length, and π_j for $\langle pc_j, \sigma_j \rangle$.

Traces

Games In Progress



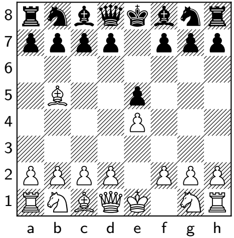
(a) $\langle \mathit{Black}, \sigma_1 \rangle$ (b) $\langle \mathit{White}, \sigma_2 \rangle$ (c) $\langle \mathit{Black}, \sigma_3 \rangle$

Figure: A trace of length 3.

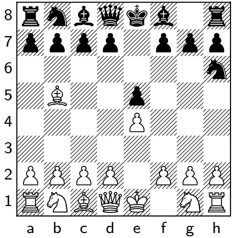
- A **trace** is a path that starts from an initial configuration.
  - In particular, a trace is a path that is subject to the additional requirement $\sigma_{init} \rightarrow_{Chess} \sigma_1$.
- Note that the chess short-hand (pictured above the boards) is simply an alternative way to describe a trace.

Extending a Path

Making a Move



(a) After a path π



(b) After $\pi \frown \langle White, \sigma_4 \rangle$

- The act of **extending** a path by transitioning from its last state is written $\pi \frown \langle pc, \sigma \rangle$.

Prefixes and Suffixes

"Halves" of In-Progress Chess Games

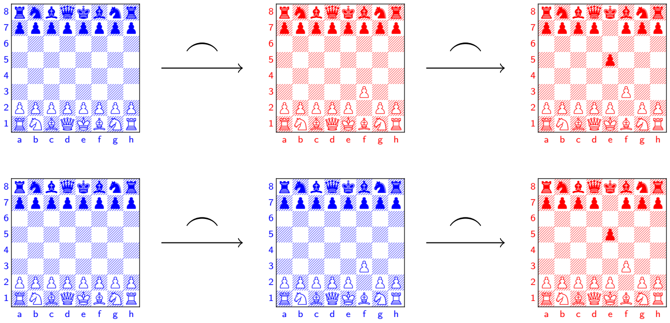
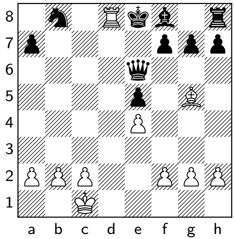


Figure: Two prefixes and suffixes of the same trace

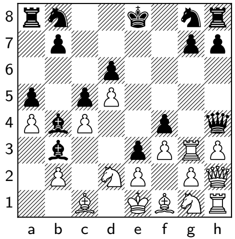
- Given a path (or trace) π:
  - Any subsequence $\pi_1 \ldots \pi_j$ is called a **prefix**.
  - Any subsequence $\pi_i \ldots \pi_n$ (where $n = |\pi|$) is called a **suffix**.

Termination of Traces

End of a Chess Game



(a) Checkmate

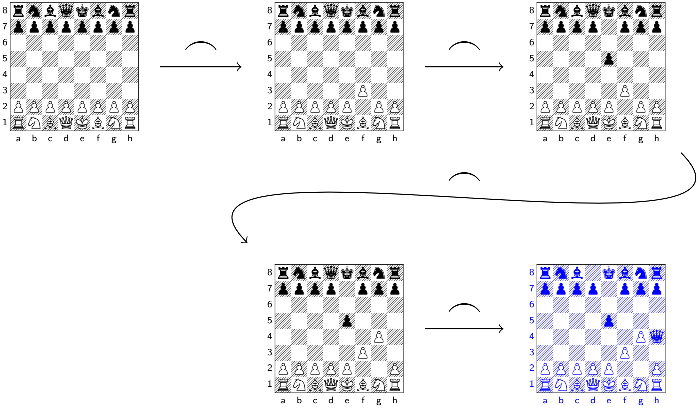


(b) Draw (Stalemate)

- Circumstances wherefrom no legal state transition (move) follows are called **final states**. In chess:
  1. Checkmate: game ends; one player wins.
  2. Draw: game ends; neither player wins. E.g.:
     - 2.1 Stalemate: player has no legal move.
     - 2.2 Threefold repetition: board in same configuration three times.
     - 2.3 Fifty-move rule: no capture or pawn move in the last 50 moves

Maximal Traces

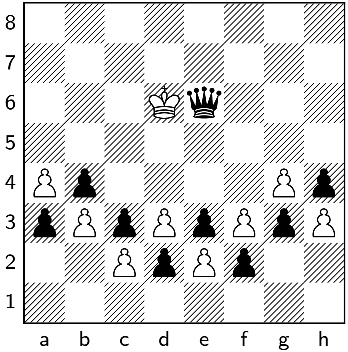
Complete Games



- A trace that ends in a final state is called **maximal**.

Trace Properties

Questions about Chess Games



- We can ask a number of questions, about all possible games of chess (**universal properties**), or about specific games (**existential properties**).
- For example, is it possible to reach the above board?
  - This is a **reachability** question.

Trace Properties

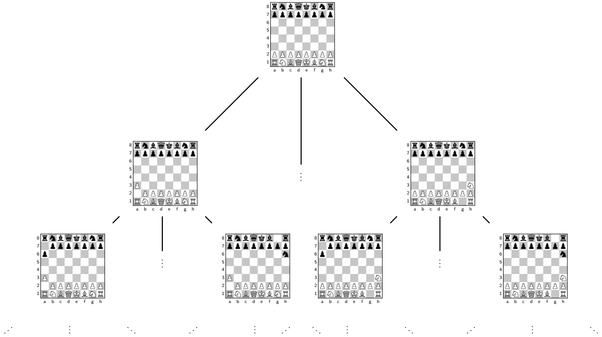
Further Questions about Chess Games

?

- What is the shortest (or longest) checkmate?
- Given a board:
  - What sequence of moves might have lead there?
  - What sequence of moves might have lead there, at a given move number?
  - Is it possible that a given player wins?
  - Must a given player always win?
  - Which positions, reachable from that configuration, lead to a given player always winning?

Trace Properties

Answers via the Execution Tree

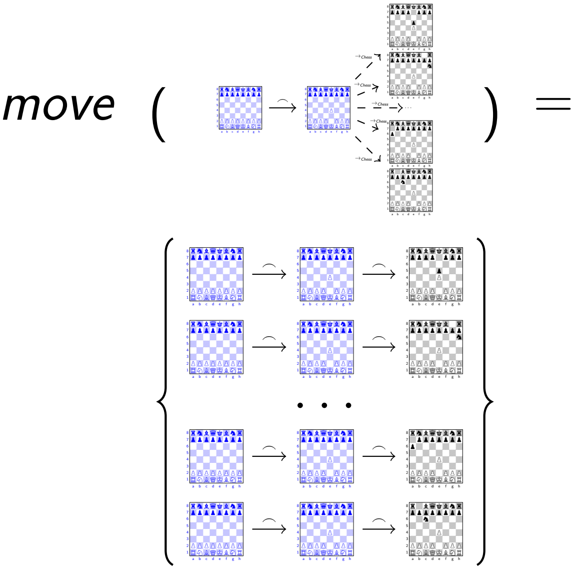


- In order to answer these questions, we need to somehow create an object that contains every chess game.
- Hence, we will now construct the **execution tree** for chess.

The Trace Extension Operator, *move*(π)

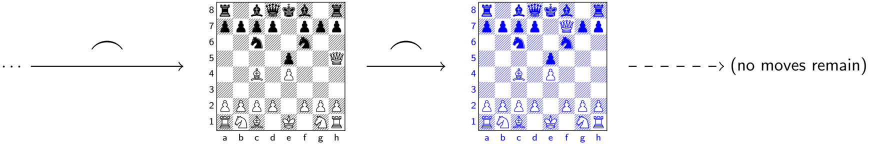
- The **trace extension operator**, *move*(π), takes as input a trace π, and produces as output all traces where one legal state transition was made from the end of π.
- Formally: $move(\pi) = \{\pi \frown \sigma \mid \sigma \in post_{\rightarrow_{Chess}}(\pi_i), |\pi| = i\}$

The Trace Extension Operator, *move*(π)



- The first boards represent the trace π.

$move(\pi)$ and Termination



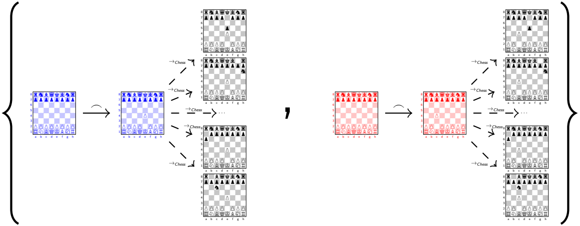
- When a trace π_{final} has reached a final state, i.e., a checkmate or draw position, no legal moves follow.
- Therefore, $move(\pi_{final})$ produces no traces, i.e. it returns the empty set $\emptyset$.

$move_{\wp}(\{\pi_i\})$, the Pointwise Extension of $move(\pi)$

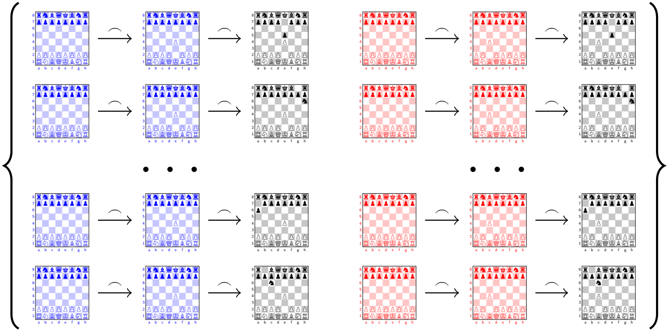
- The **pointwise-extended trace extension operator**, $move_{\wp}(\pi)$, takes as input *one or more traces* $\{\pi_i\}$, and outputs the result of applying $move(\pi_j)$ for each j.
  - I.e., it is simply $move(\pi)$ applied to one or more traces.
- Formally: $move_{\wp}(X) = \bigcup_{\pi \in X} move(\pi)$
- For simplicity, in the rest of the presentation, we will simply write *move* in place of $move_{\wp}$.
  - Slovenly, but less notation to remember.

$move_{\wp}(\{\pi_1, \pi_2\})$

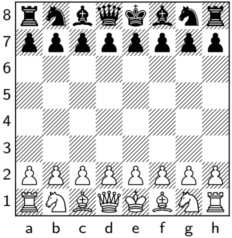
$$move_{\wp}\Big($$



$$\Big) =$$

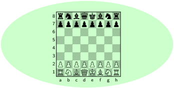


T_i, All Traces of Length i

- Write T_i to denote the set of traces of length i.
- $T_1 := \Big\{ \Big\langle$ White, 

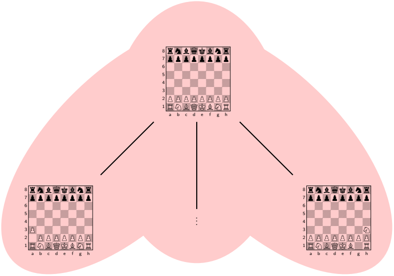
$\Big\rangle \Big\}$.
  - I.e., the initial board before any moves have been made.
- Now $T_k = move(T_{k-1})$ for $k \geq 2$.
  - I.e., all traces of length k.
  - $T_2 = move(T_1)$
  - $T_3 = move(T_2) = move(move(T_1)) = move^2(T_1)$
  - ...

$move(T_k)$ and T_{k+1}, Visualized



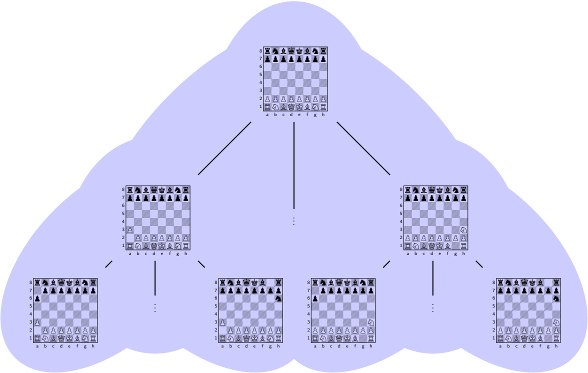
- T_1, all traces of length 1.

$move(T_k)$ and T_{k+1}, Visualized



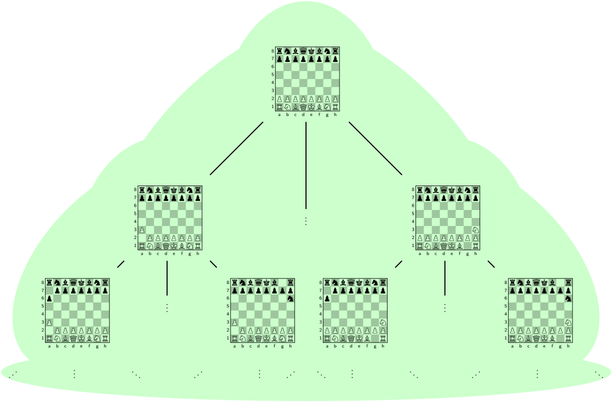
- T_1, all traces of length 1.
- $T_2 = move(T_1)$, all traces of length 2.

$move(T_k)$ and T_{k+1}, Visualized



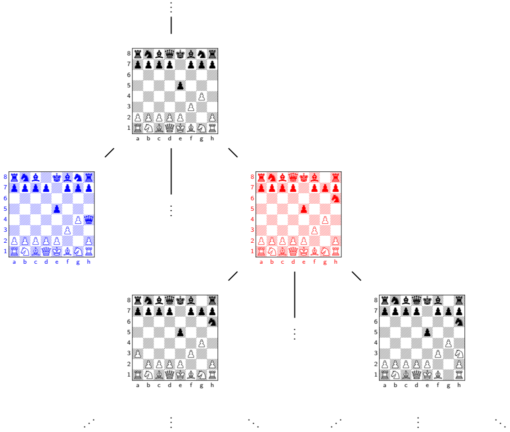
- T_1, all traces of length 1.
- $T_2 = move(T_1)$, all traces of length 2.
- $T_3 = move(T_2)$, all traces of length 3.

$move(T_k)$ and T_{k+1}, Visualized



- T_1, all traces of length 1.
- $T_2 = move(T_1)$, all traces of length 2.
- $T_3 = move(T_2)$, all traces of length 3.
- $T_4 = move(T_3)$, all traces of length 4.

$move(T_k)$, T_{k+1}, and Termination



- When a trace reaches a final state, the branch stops growing.
- Black has checkmated down the left branch. Thus, no traces extend on the left.
- The right branch does not correspond to a final state, and so continues extending.

$T_{\leq k}$, All Traces up to k in Length

- We define $T_{\leq k}$ to be the set of traces of length k or less.
- Formally, $T_{\leq k} = \bigcup_{i=1}^{k} T_i$.

Value	Result	Description
$T_{\leq 1}$	T_1	The initial board
$T_{\leq 2}$	$T_1 \cup T_2$	The initial board plus all traces of length 2
$T_{\leq 3}$	$T_1 \cup T_2 \cup T_3$	The initial board plus all traces of lengths 2 and 3
...	...	...
$T_{\leq j}$	$T_1 \cup T_2 \cup \cdots \cup T_{j-1} \cup T_j$	All traces of length up to j

$traces^j(\emptyset)$, a Generator for $T_{\leq j}$

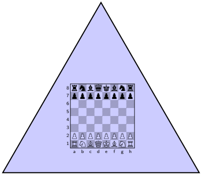
- Define a function, $traces(X) = move(X) \cup T_1$.
- Write $traces^k(X)$ for $traces(traces^{k-1}(X))$.
- Now examine $traces^k(\emptyset)$ for various k.

Table: The **iterates** of $traces^j(\emptyset)$

Value	Result
$traces^1(\emptyset)$	$T_{\leq 1}$
$traces^2(\emptyset)$	$T_{\leq 2}$
$traces^3(\emptyset)$	$T_{\leq 3}$
...	...
$traces^j(\emptyset)$	$T_{\leq j}$

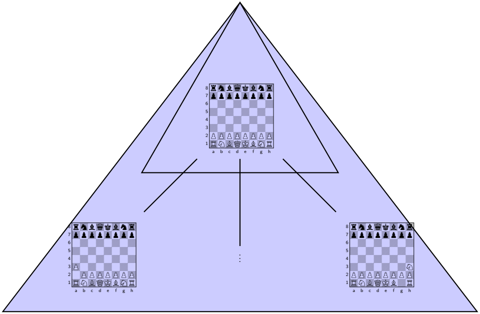
- Hence $traces^j(\emptyset)$ **generates** $T_{\leq j}$.

$T_{\leq k}$ and $traces^k(\emptyset)$, Visualized



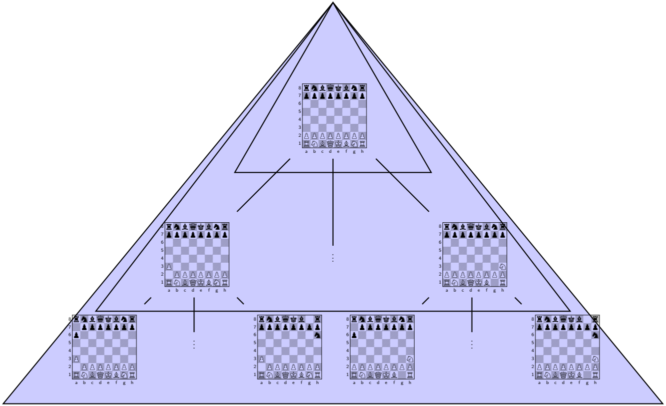
- $T_{\leq 1} = traces^1(\emptyset)$, all traces up to length 1.

$T_{\leq k}$ and $traces^k(\emptyset)$, Visualized



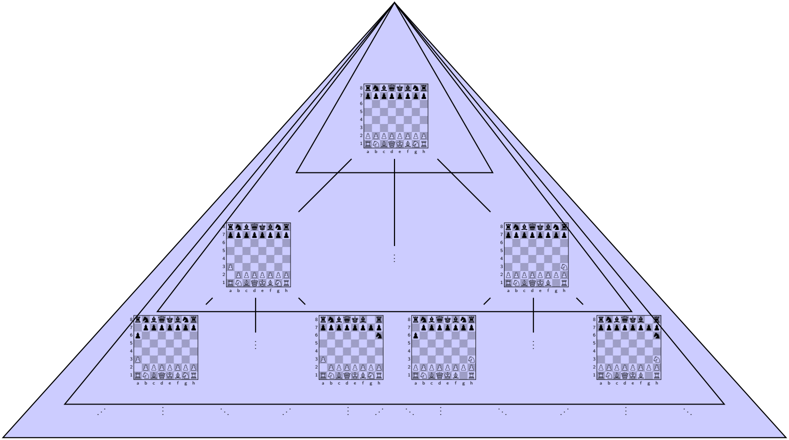
- $T_{\leq 1} = traces^1(\emptyset)$, all traces up to length 1.
- $T_{\leq 2} = traces^2(\emptyset)$, all traces up to length 2.

$T_{\leq k}$ and $traces^k(\emptyset)$, Visualized



- $T_{\leq 1} = traces^1(\emptyset)$, all traces up to length 1.
- $T_{\leq 2} = traces^2(\emptyset)$, all traces up to length 2.
- $T_{\leq 3} = traces^3(\emptyset)$, all traces up to length 3.

$T_{\leq k}$ and $\mathit{traces}^k(\emptyset)$, Visualized



- $T_{\leq 1} = \mathit{traces}^1(\emptyset)$, all traces up to length 1.
- $T_{\leq 2} = \mathit{traces}^2(\emptyset)$, all traces up to length 2.
- $T_{\leq 3} = \mathit{traces}^3(\emptyset)$, all traces up to length 3.
- $T_{\leq 4} = \mathit{traces}^4(\emptyset)$, all traces up to length 4.

$T_{\leq k}$, $traces^k(\emptyset)$, and Termination

The Execution Tree

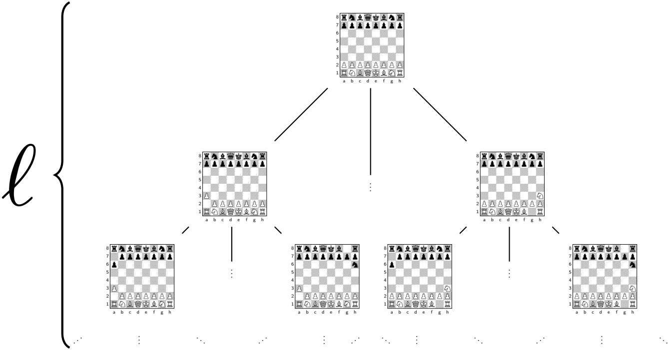
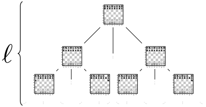


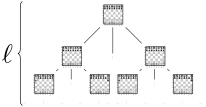
Figure: The **execution tree**, of height ℓ

- Due to the laws of chess, all games terminate.
- In particular, there is a longest chess game, say of length ℓ.
- Therefore, $T_{\leq \ell} = traces^{\ell}(\emptyset)$ contains all possible chess games.
- The tree in which every branch has reached a terminal state is called the **execution tree**.

$T_{\leq k}$, $traces^k(\emptyset)$, and Termination

Fixed Points

$traces($ 

$) =$ 

- Because terminated traces cannot be extended, we have that:
  - $T_{\leq \ell+1} = traces^{\ell+1}(\emptyset) =$
  - $traces(traces^{\ell}(\emptyset)) =$
  - $traces(T_{\leq \ell}) =$
  - $T_{\leq \ell}$
- In conclusion, $traces(T_{\leq \ell}) = T_{\leq \ell}$.
- For a function f, a value x such that $f(x) = x$ is called a **fixed point** of f.
- Hence, $T_{\leq \ell}$ is a fixed point of *traces*.

The Least Fixedpoint Trace Semantics

Definition and Notation

$$\mathbf{S}[\![\mathit{Chess}]\!] \;=\; \mathbf{lfp}^{\subseteq}\, \mathit{traces} \;=\; T_{\leq \ell}$$

- $T_{\leq \ell}$ is the *smallest* set that is a fixed point of *traces*; hence it is the **least fixed point**.
  - *Smallest* as in, if $S = \mathit{traces}(S)$, then $T_{\leq \ell} \subseteq S$.
- $T_{\leq \ell}$ is called the **least fixed point trace semantics**.
- We write $\mathbf{S}[\![\mathit{Chess}]\!] = \mathbf{lfp}^{\subseteq}\, \mathit{traces} = T_{\leq \ell}$.
  - $\mathbf{S}[\![\mathit{Chess}]\!]$: the **semantics of chess**, also called the **set of traces**.
  - $\mathbf{lfp}^{\subseteq}\, \mathit{traces}$: the least fixed point of the *traces* function.

The Least Fixedpoint Trace Semantics

Computation

We can compute the least fixed point of *traces* as follows:

1. $Traces := \emptyset$
2. $Traces_{new} := traces(Traces)$
3. If $Traces_{new} \neq Traces$ then
   - $Traces := Traces_{new}$
   - Go to 2
4. Return $Traces$

Note that this computation does not depend upon knowing what ℓ is; it terminates after ℓ steps automatically.

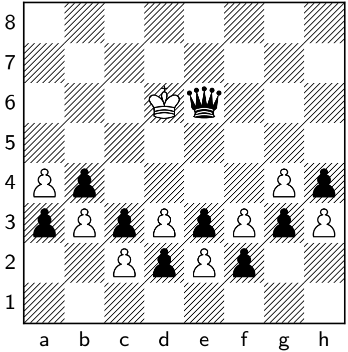
The Maximal Trace Semantics

Set of all Terminated Chess Games

$$\mathbf{S}_{Max}[\![Chess]\!]$$

- The **maximal trace semantics** $\mathbf{S}_{Max}[\![Chess]\!]$ is a subset of $\mathbf{S}[\![Chess]\!]$ that contains only the maximal (terminated) traces.
  - $\mathbf{S}[\![Chess]\!]$ also contains all prefixes of maximal traces.
- Formally: $\mathbf{S}_{Max}[\![Chess]\!] = \{\, T \mid T \in \mathbf{S}[\![Chess]\!], T \text{ is maximal } \}$

Deciding Trace Properties



- Now that we have constructed $\mathbf{S}[\![Chess]\!]$ and $\mathbf{S}_{Max}[\![Chess]\!]$, we can use them to answer the questions about traces that we have previously posed.

Deciding Trace Properties

Average Game Length

$$\frac{\displaystyle\sum_{\pi \in \mathbf{S}_{Max}[\![Chess]\!]} |\pi|}{|\mathbf{S}_{Max}[\![Chess]\!]|}$$

- To find the average game length, we simply divide the sum of the lengths of the individual games by the number of games.
- Similarly, we can find the length of the average checkmate or draw by restricting our attention to those games only.

Deciding Trace Properties

Lengths of Checkmates

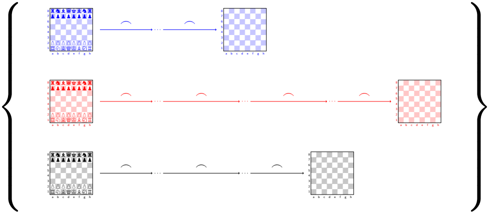


Figure: The shortest and longest of a set of checkmating traces

Shortest and longest checkmate:

- $Checkmate = \{\, T \mid T \in \mathbf{S}_{Max}[\![Chess]\!],\ T \text{ checkmates } \}$.
  - I.e., the set of all games that checkmate.
- $Check_{min} = \min_{T \in Checkmate} |T|$
- $Check_{max} = \max_{T \in Checkmate} |T|$

Deciding Trace Properties

Reachability at a Move

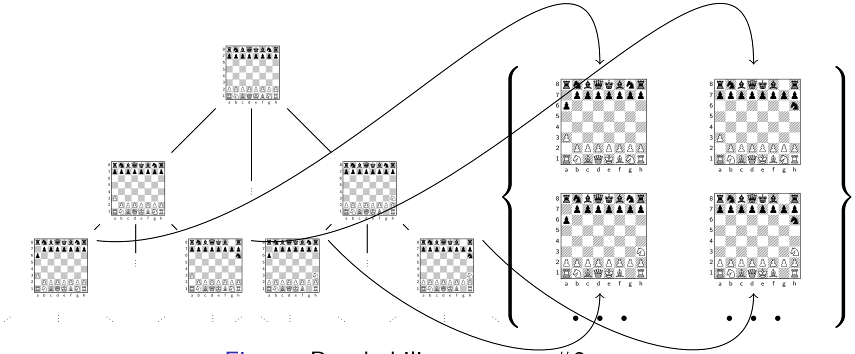


Figure: Reachability at move #2.

A board b is reachable at move $\#n$ if
$b \in \{\sigma_n \mid \sigma_1 \ldots \sigma_n \in \mathbf{S}[\![Chess]\!]\}$.

Deciding Trace Properties

Reachability Generally

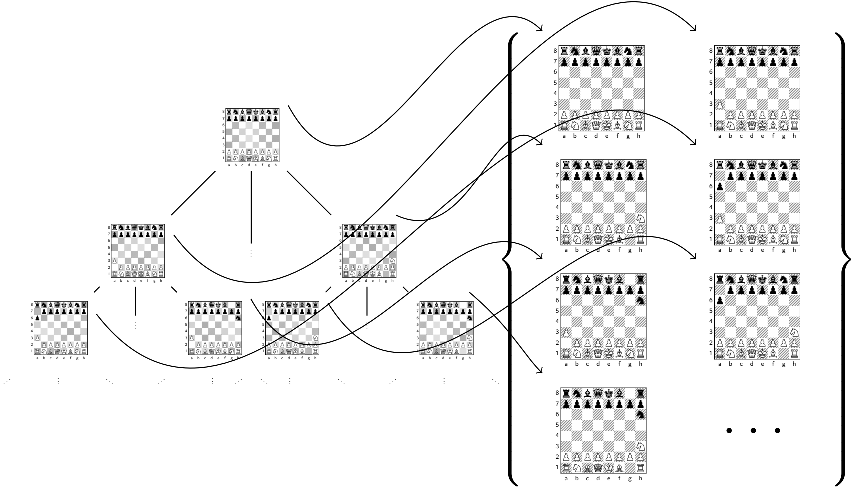
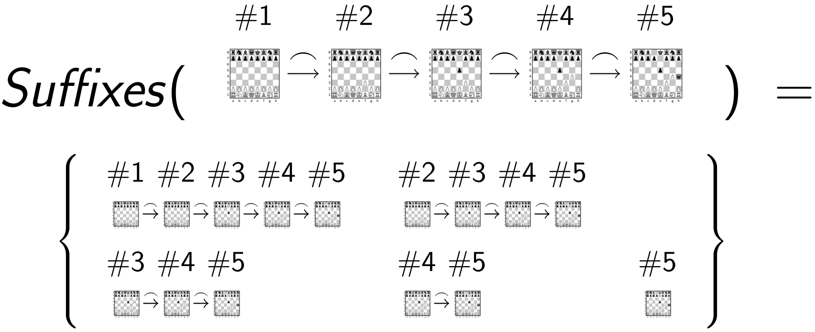


Figure: Reachability at any move.

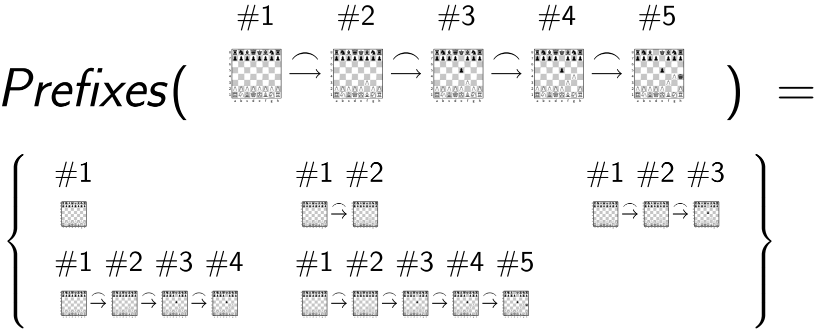
A board b is reachable if $b \in \{\sigma_j \mid \sigma_1 \ldots \sigma_j \in \mathbf{S}[\![Chess]\!]\}$.

All Suffixes



- $\mathit{Suffixes}(\pi) = \{\pi_i \ldots \pi_n \mid 1 \leq i \leq |\pi|, |\pi| = n\}$
  - All suffixes from one trace.
- $\mathit{Suffixes}_{\wp}(X) = \bigcup_{\pi \in X} \mathit{Suffixes}(\pi)$
  - All suffixes from a set of traces; called the **pointwise extension** of the *Suffixes* function.

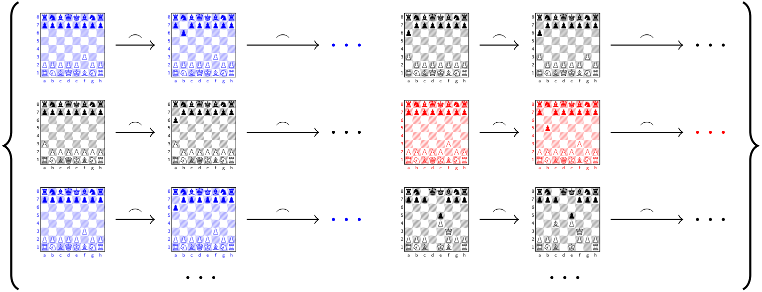
All Prefixes



- $\mathit{Prefixes}(\pi) = \{\pi_1 \ldots \pi_i \mid 1 \leq i \leq |\pi|\}$
  - All prefixes from one trace.
- $\mathit{Prefixes}_{\wp}(X) = \bigcup_{\pi \in X} \mathit{Prefixes}(\pi)$
  - All prefixes from a set of traces; called the **pointwise extension** of the *Prefixes* function.

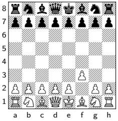
All Suffixes of Complete Games

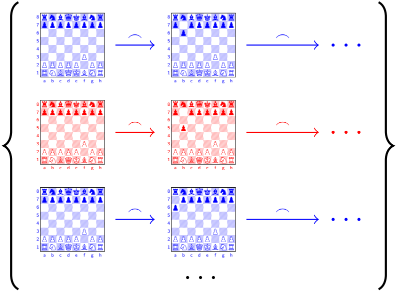
$$\mathit{Suffixes}_{\wp}\ \left(\ \mathbf{S}_{\mathit{Max}}[\![\mathit{Chess}]\!]\ \right) =$$



- All suffixes of complete games.

All Maximal Suffixes Beginning with a Board

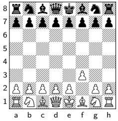
$$\mathit{SuffixBeginsWith}\ ($$  $$) =$$



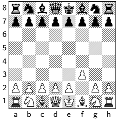
- All maximal games beginning with some board.

Deciding Trace Properties

White Can Win, an Existential Property

$CanWinFrom$ (


) $=$

$\exists \pi \in$ $SuffixBeginsWith$ (


) $\cdot$

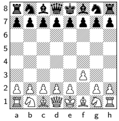
$Wins$ (π , $White$) ?

- The formula is true if there exists a complete game beginning from the indicated board where *White* wins.

Deciding Trace Properties

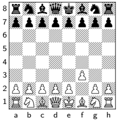
White Always Wins, a Universal Property

AlwaysWinFrom (



) =

$\forall \pi \in$ *SuffixBeginsWith* (



) $\cdot$

Wins (π , *White*) ?

- The formula is true if for all complete games beginning from the indicated board, *White* wins.

Deciding Trace Properties

Can Reach a Position Where One Player Always Wins

$$\mathit{CanAlwaysWin}\ (\ \pi\ ) =$$

$$\exists \varpi \in\ \mathit{Reachable}\ (\ \pi\ )\ \cdot$$

$$\mathit{AlwaysWinFrom}\ (\ \varpi\ ,\ \mathit{White}\ )\ ?$$

- The formula is true if, given a game-in-progress π, there exists some reachable board ϖ such that *White* always wins.