CSCI 5535 Fundamentals of Programming Languages

Lec 1: Introduction

CUPLV

CU Programming Languages
& Verification

About me

Gowtham Kaki

- Assistant Professor, Dept. of Computer Science
- New to CU Boulder - Joined Fall 2020
  - This is my first in-person class!
- Research: Programming Languages and Formal Methods. Applications in Concurrent and Distributed Systems.
- Best known for Quelea (PLDI 2015) and MRDTs (OOPSLA 2019).
- In free time: biking (recently bought a Cannondale Trail 8!) , reading (pop-science is my thing), and strolling aimlessly.

About me

Gowtham Kaki

Pronounced

g-OW-thum

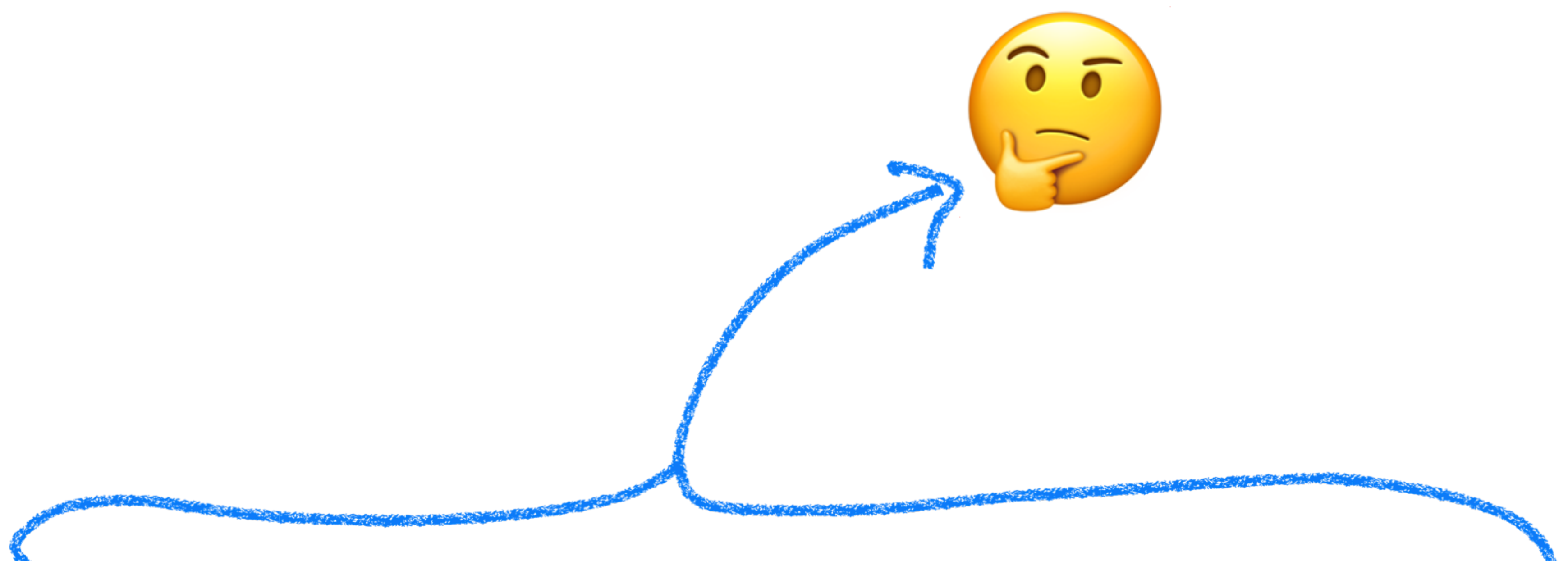
close enough!

- Assistant Professor, Dept. of Computer Science
- New to CU Boulder - Joined Fall 2020
  - This is my first in-person class!
- Research: Programming Languages and Formal Methods. Applications in Concurrent and Distributed Systems.
- Best known for Quelea (PLDI 2015) and MRDTs (OOPSLA 2019).
- In free time: biking (recently bought a Cannondale Trail 8!) , reading (pop-science is my thing), and strolling aimlessly.

About CSCI 5535 / ECEN 5533

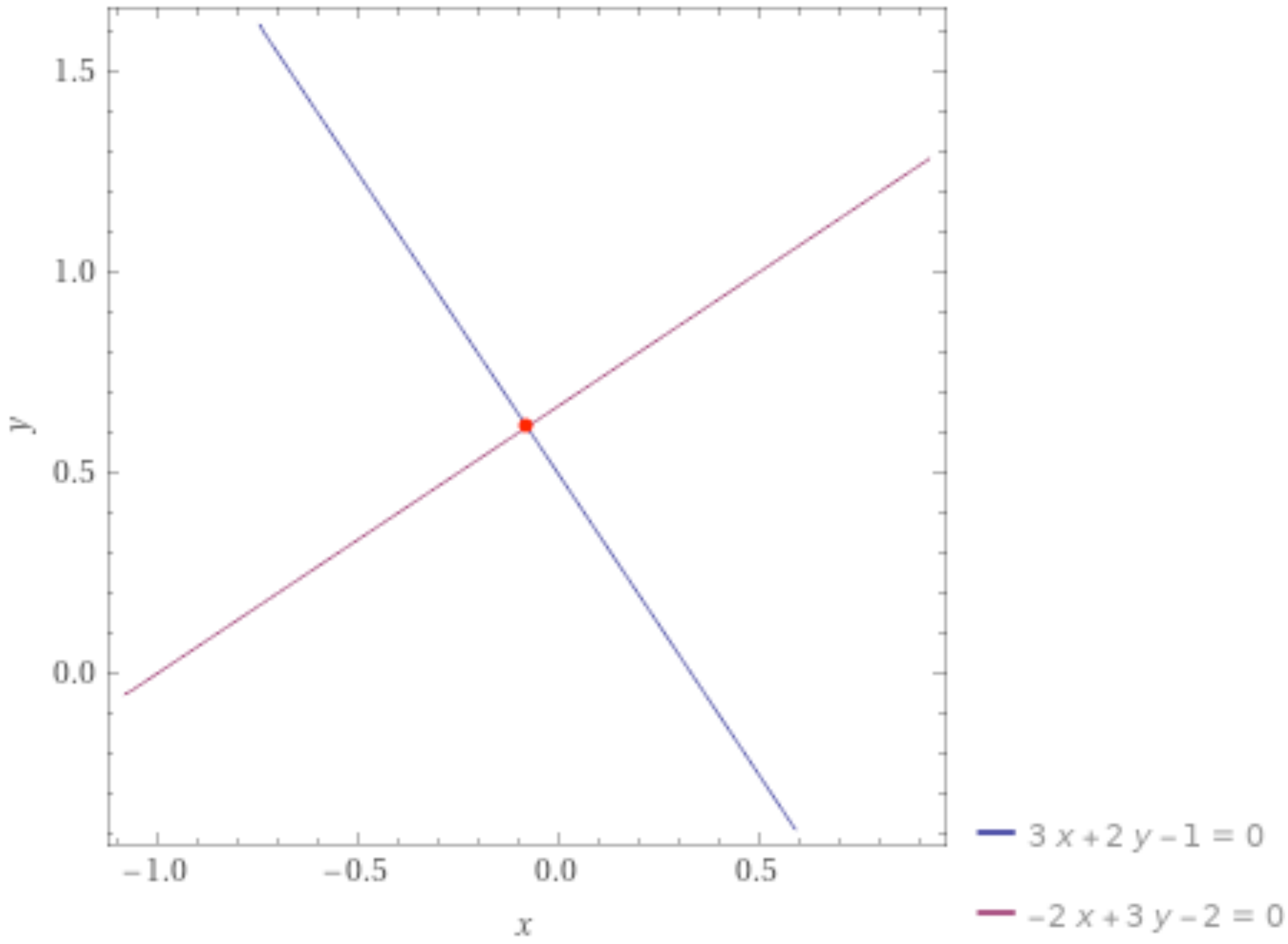
Mathematical foundations of computer programs and programming languages.

About CSCI 5535 / ECEN 5533



Mathematical foundations of computer programs and programming languages.

Recall High-School Algebra ...



- Consider the equations:

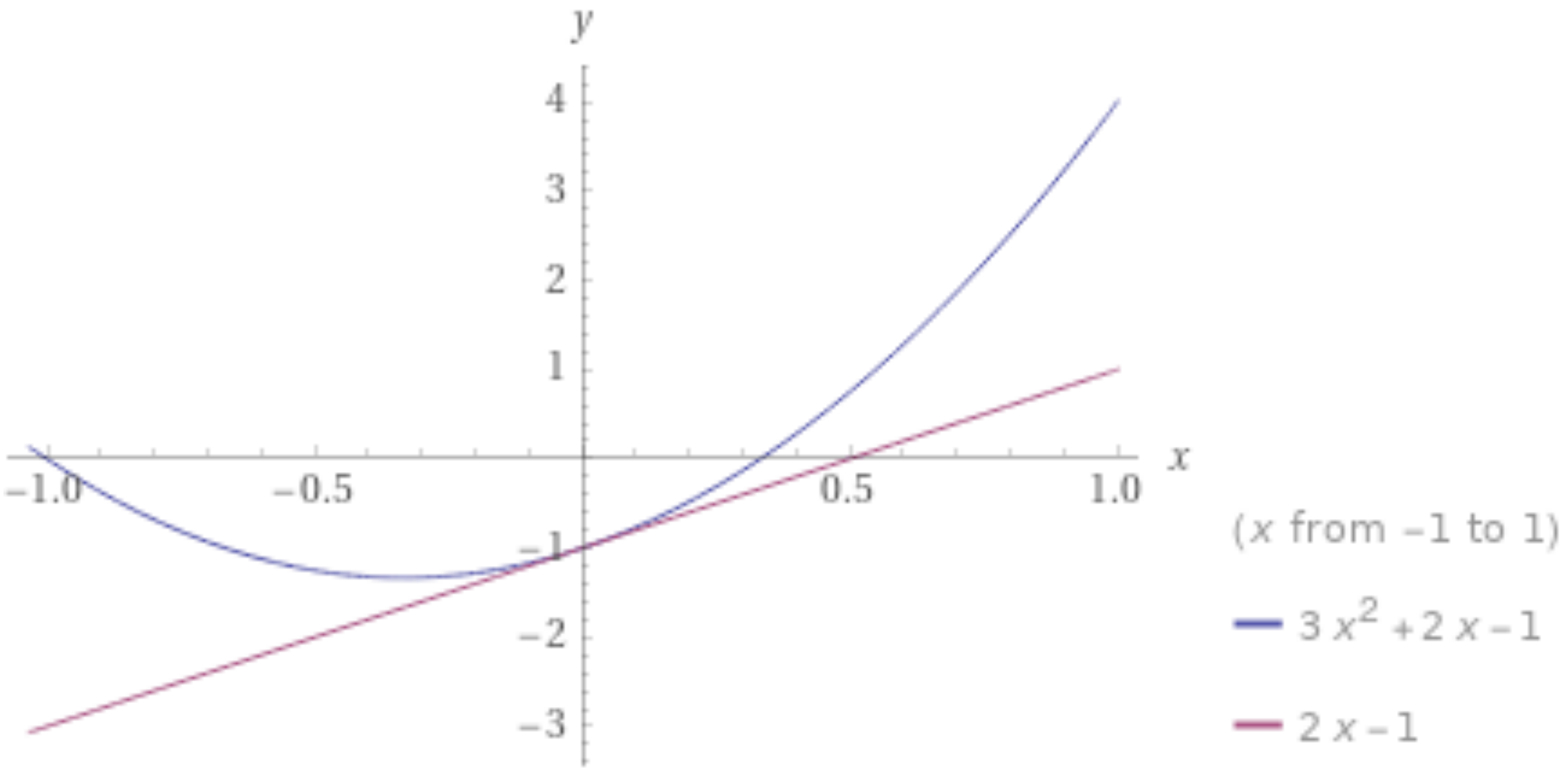
$$3x + 2y - 1 = 0$$

$$-2x + 3y - 2 = 0$$

- Different but not *fundamentally* different.

- Different instantiations of $ax + by + c = 0$

Recall High-School Algebra ...



- Now consider the equations:

$$y = 3x^2 + 2x - 1$$

$$y = 2x - 1$$

- Fundamentally different equations.
  - One is quadratic, other is linear.
- $y = ax^2 + bx + c$ is more *expressive* / *powerful* than $ax + by + c = 0$

Are computer programs analogous to algebraic functions?

C

```c
f(n){
    return n<4?1:f(--n)+f(--n);
}
main(a,b){
    for(scanf("%d",&b);a++<=b;printf("%d ",f(a)));
}
```

Java

```java
import java.io.*;
public class Fib
{
    public static void main(String args[]) throws IOException
    {
        int n,f1,f2,f3;
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        n = Integer.parseInt(br.readLine());
        f1=0;
        f2=1;
        if(n>0)
        {
            for(int i=0; i<n; i++)
            {
                System.out.println(" "+f1);
                f3=f1+f2;
                f1=f2;
                f2=f3;
            }
        }
    }
}
```

Are computer programs analogous to algebraic functions?

C

```c
f(n){
    return n<4?1:f(--n)+f(--n);
}
main(a,b){
    for(scanf("%d",&b);a++<=b;printf("%d ",f(a)));
}
```

$\overset{?}{\cong}$

Java

```java
import java.io.*;
public class Fib
{
    public static void main(String args[]) throws IOException
    {
        int n,f1,f2,f3;
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        n = Integer.parseInt(br.readLine());
        f1=0;
        f2=1;
        if(n>0)
        {
            for(int i=0; i<n; i++)
            {
                System.out.println(" "+f1);
                f3=f1+f2;
                f1=f2;
                f2=f3;
            }
        }
    }
}
```

Are computer programs analogous to algebraic functions?

C

```
f(n){
    return n<4?1:f(--n)+f(--n);
}
main(a,b){
    for(scanf("%d",&b);a++<=b;printf("%d ",f(a)));
}
```

$\overset{?}{\cong}$

Java

```
import java.io.*;
public class Fib
{
    public static void main(String args[]) throws IOException
    {
        int n,f1,f2,f3;
        BufferedReader br =
            new BufferedReader(new InputStreamReader(System.in));
        n = Integer.parseInt(br.readLine());
        f1=0;
        f2=1;
        if(n>0)
        {
            for(int i=0; i<n; i++)
            {
                System.out.println(" "+f1);
                f3=f1+f2;
                f1=f2;
                f2=f3;
            }
        }
    }
}
```

Q. Is there a mathematics to answer such questions decisively?

A. Yes!

About CSCI 5535 / ECEN 5533

Mathematical foundations of computer programs and programming languages.

- To understand fundamental differences among various programming styles and languages.
- To learn various ways in which one can ascribe a *meaning* to a program.
- To ask precise questions about computer programs and to decisively answer them.
    - E.g: “Does this program stably sort a list of numbers?”, “Does this program ever terminate?” etc.

About CSCI 5535 / ECEN 5533

Mathematical foundations of computer programs and programming languages.

- To understand fundamental differences among various programming styles and languages.
- To learn various ways in which one can ascribe a *meaning* to a program.
- To ask precise questions about computer programs and to decisively answer them.
    - E.g: “Does this program stably sort a list of numbers?”, “Does this program ever terminate?” etc.

About CSCI 5535 / ECEN 5533

Mathematical foundations of computer programs and programming languages.

- To understand fundamental differences among various programming styles and languages.
- To learn various ways in which one can ascribe a *meaning* to a program.
- To ask precise questions about computer programs and to decisively answer them.
  - E.g: “Does this program stably sort a list of numbers?”, “Does this program ever terminate?” etc.

“Program Verification”

Prove that a program P satisfies a property φ

What is a proof?

Theorem 2.30 (Sound). *If every branch of a semantic argument proof of $I \not\models F$ closes, then F is valid.*

Completeness is more complicated. We want to show that there exists a closed semantic argument proof of $I \not\models F$ when F is valid. Our strategy is as follows. We define a procedure for applying the proof rules. When applying the quantification rules, the procedure selects values from a predetermined countably infinite domain. We then show that when some falsifying interpretation I exists (such that $I \not\models F$) our procedure constructs, *at the limit*, a falsifying interpretation. Therefore, F must be valid if the procedures actually discovers an argument in which all branches are closed. We now proceed according to this proof plan.

Let D be a countably infinite domain of values $\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \ldots$ which we can enumerate in some fixed order. Start the semantic argument by placing $I \not\models F$ at the root and marking it as *unused*. Now assume that the procedure has constructed a partial semantic argument and that each line is marked as either *used* or *unused*. We describe the next iteration.

Select the earliest line $L : I \models G$ or $L : I \not\models G$ in the argument that is marked *unused*, and choose the appropriate proof rule to apply according to the root symbol of G's parse tree. To apply a rule, add the appropriate deductions at the end of every open branch that passes through line L; mark each new deduction as *unused*; and mark L as *used*. The application of the negation rules and the first conjunction rule is then straightforward. Applying the second (branching) conjunction rule introduces a fork at the end of every open branch, doubling the number of open branches. In applying the second quantification rule, choose the next domain element v_i that does not appear in the semantic argument so far. For the first quantification rule, assume that G has the form $\forall x.\ H$. Choose the first value v_i on which $\forall x.\ H$ has not been instantiated in any ancestor of L. Additionally, consider $I \models G$ as a second "deduction" of this rule (so that both $I \triangleleft \{x \mapsto \mathsf{v}_i\} \models H$ and $I \models G$ are added to every branch passing through L and marked as *unused*). This trick guarantees that x of $\forall x.\ H$ is instantiated on every domain element without preventing the rest of the proof from progressing. Finally, close any branch that has a contradiction resulting from a deduction in this iteration.

Proofs we are used to:

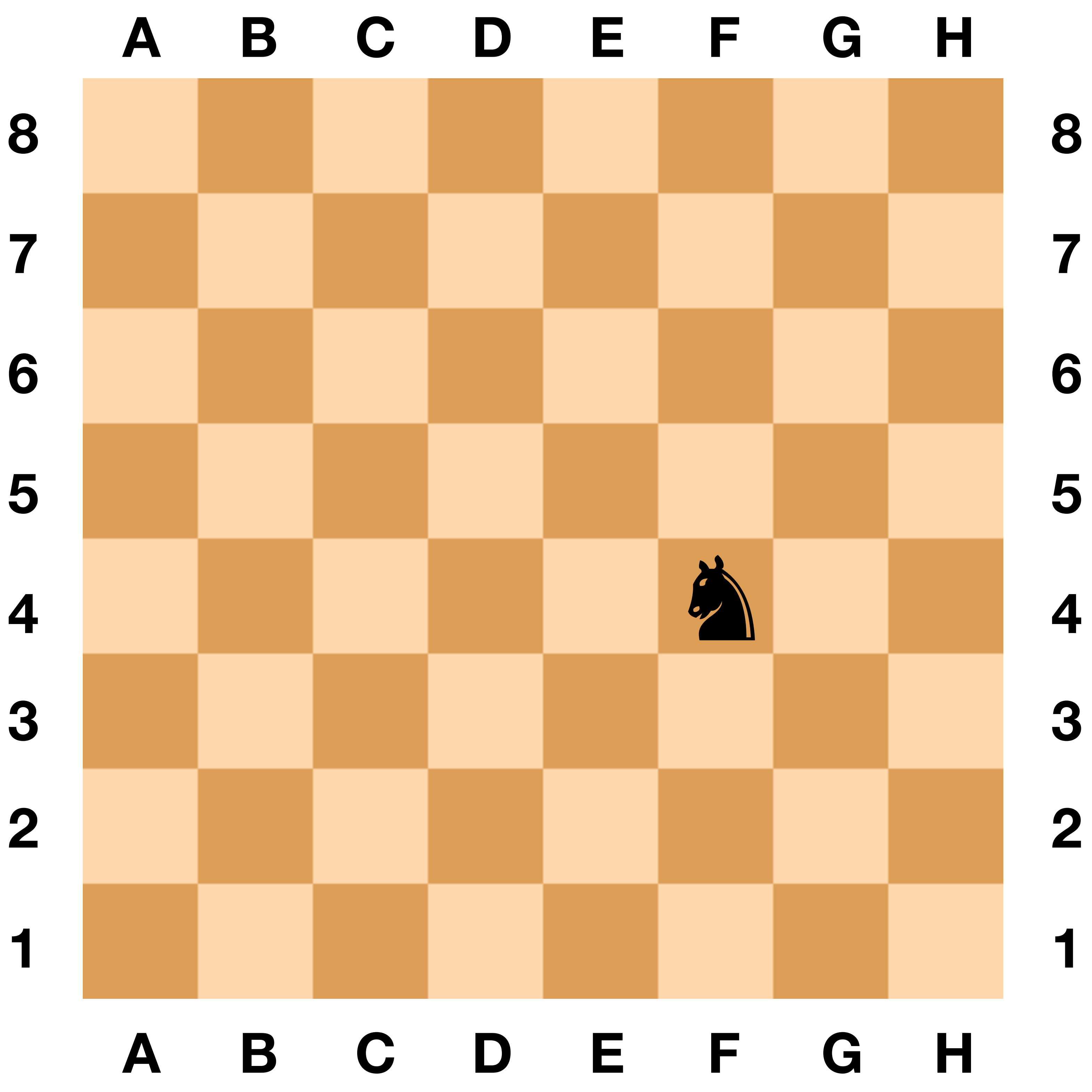
- Informal arguments
- Wall of text
- Error-prone
- Often incomprehensible.

Vs

Proofs in this class:

- Chain of precise deductions from first principles.
- Machine-checkable

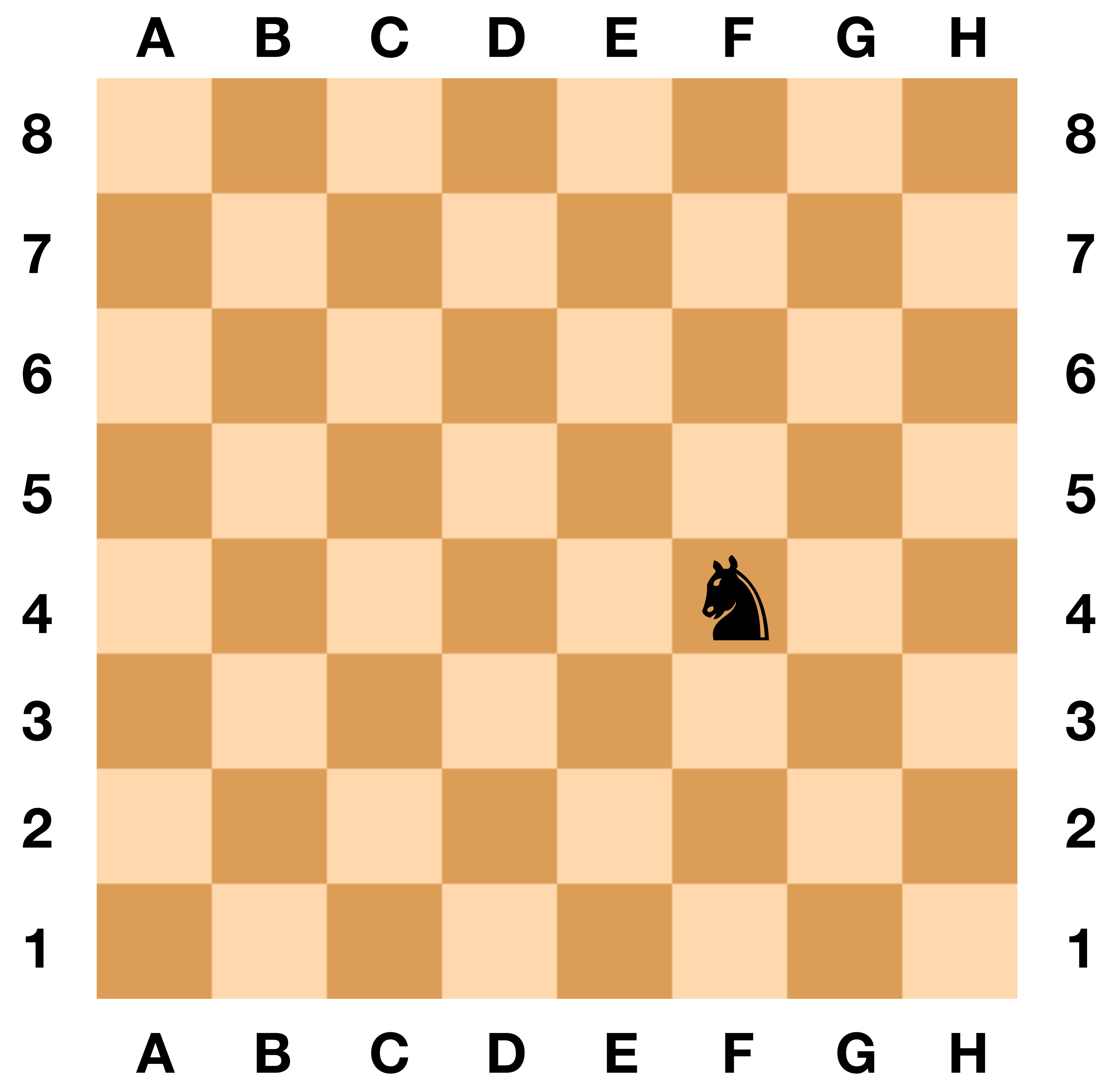
What is a proof?



Definition: $CR(i, j)$: Knight CanReach the square $\langle i, j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

What is a proof?



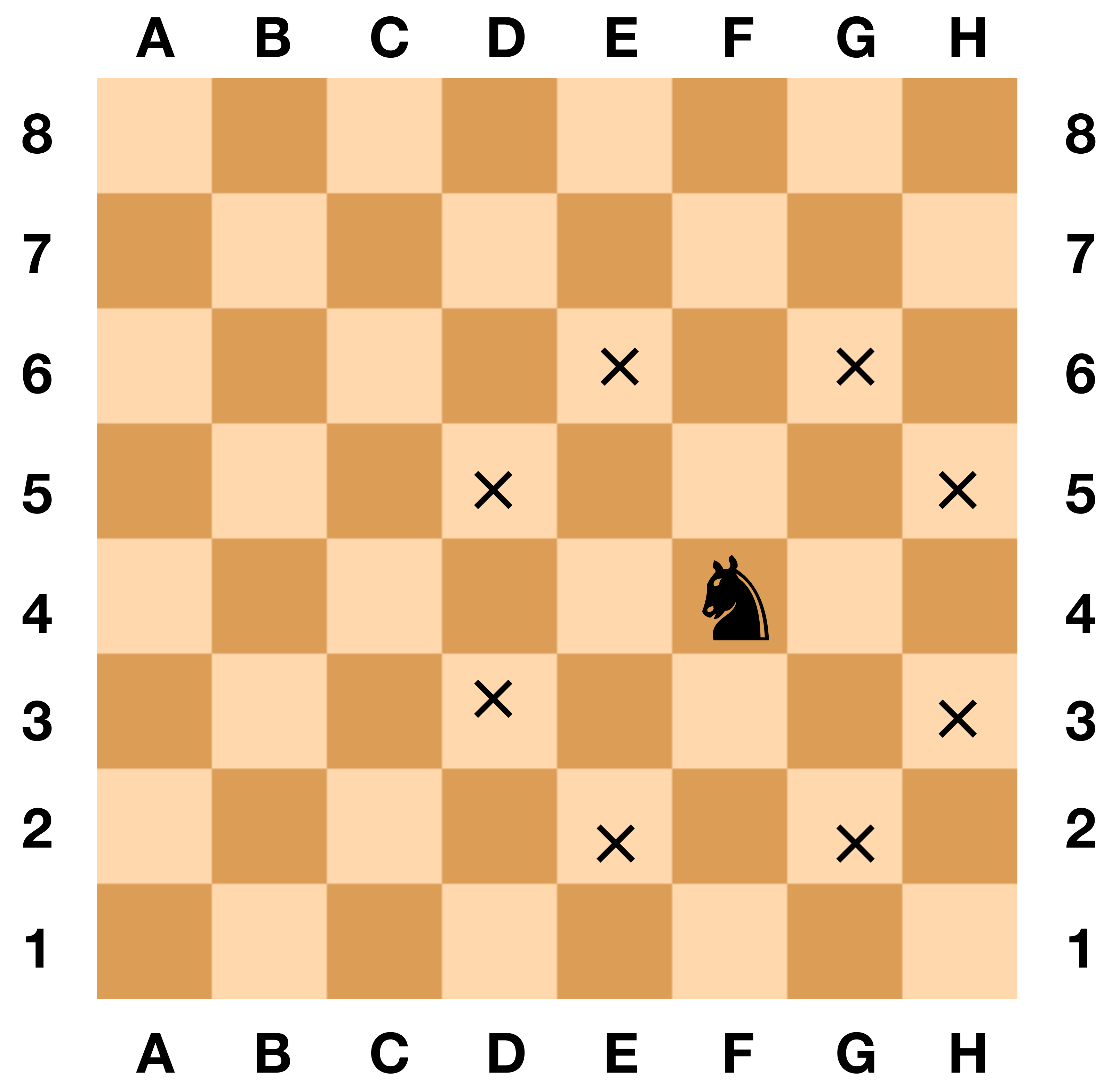
Definition: $CR(i,j)$: Knight CanReach the square $\langle i,j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

$\Downarrow$ *NamePremise* P_1

$\langle P_1 : CR(A,8) \rangle \quad \vdash \quad CR(F,4)$

What is a proof?



Definition: $CR(i,j)$: Knight CanReach the square $\langle i,j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

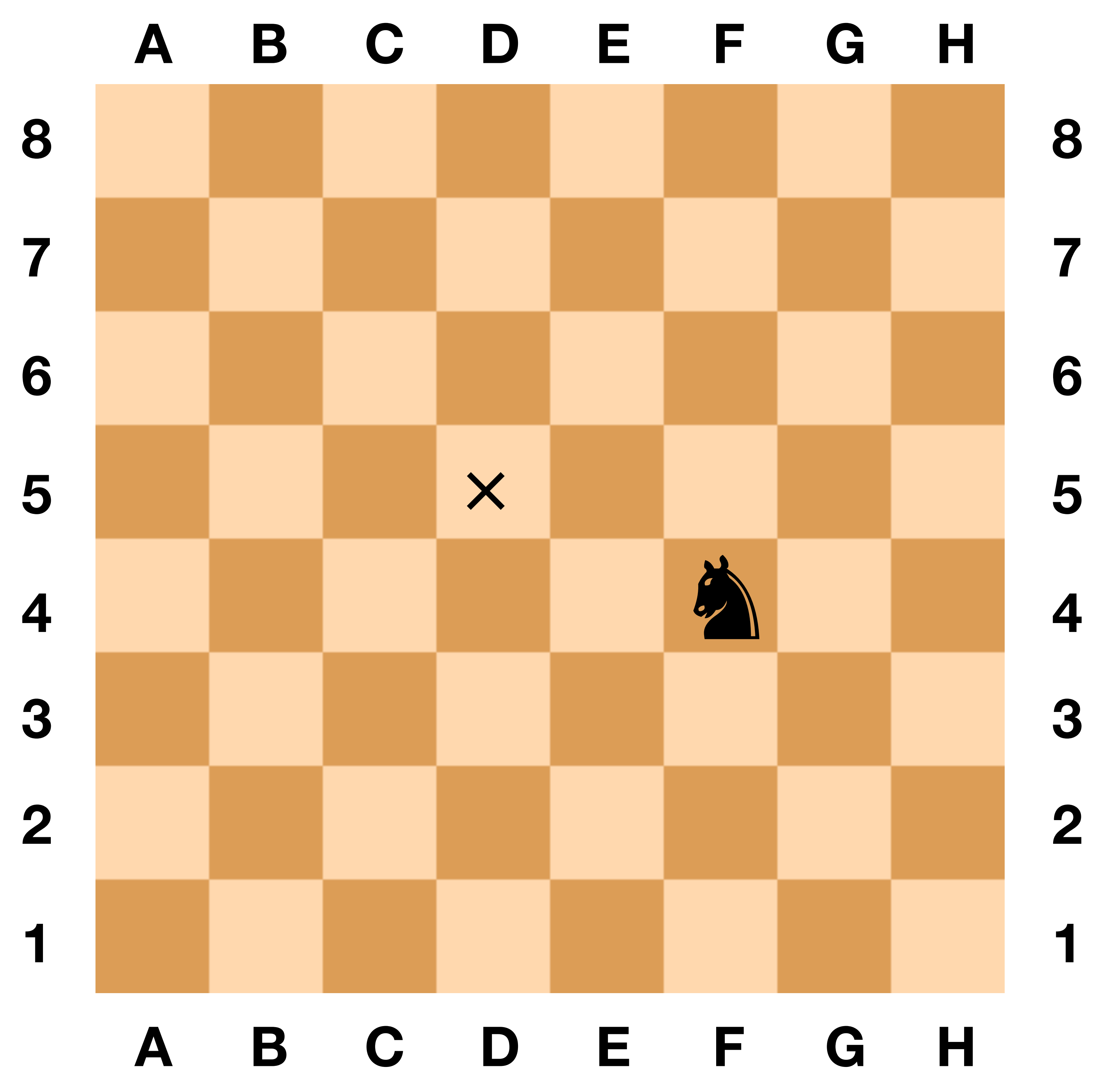
$$\Downarrow \textit{NamePremise } P_1$$

$$\langle P_1 : CR(A,8) \rangle \quad \vdash \quad CR(F,4)$$

$$\Downarrow \textit{Invert } CR(F,4)$$

$$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5) \vee CR(E,6) \vee CR(G,6) \vee CR(H,5) \vee CR(H3) \vee CR(G2) \vee CR(E,2) \vee CR(D3)$$

What is a proof?



Definition: $CR(i, j)$: Knight CanReach the square $\langle i, j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

$\Downarrow$ *NamePremise* P_1

$\langle P_1 : CR(A,8) \rangle \vdash CR(F,4)$

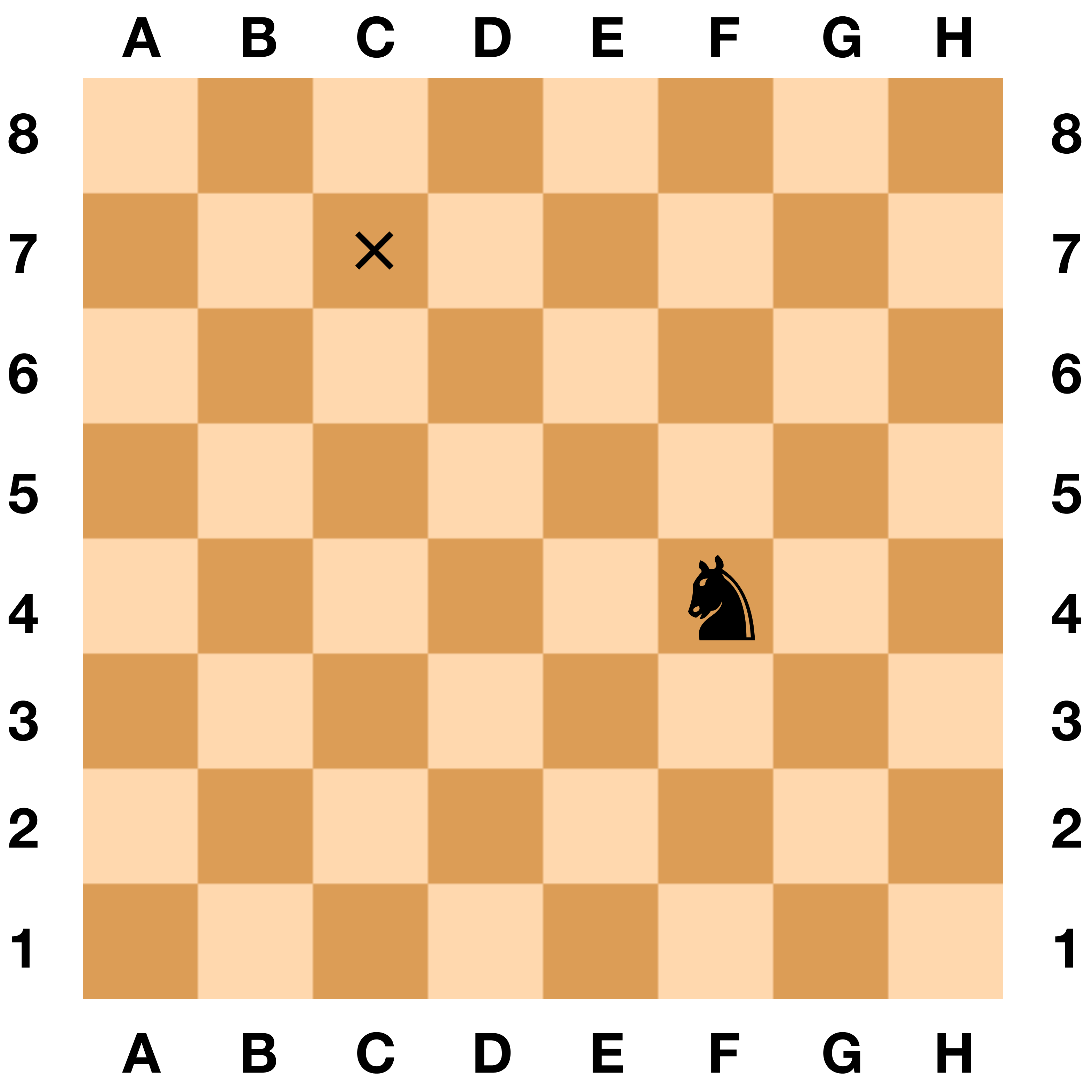
$\Downarrow$ *Invert* $CR(F,4)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5) \vee CR(E,6) \vee CR(G,6) \vee CR(H,5)$
$\vee CR(H3) \vee CR(G2) \vee CR(E,2) \vee CR(D3)$

$\Downarrow$ *PickDisjunct* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5)$

What is a proof?



Definition: $CR(i, j)$: Knight CanReach the square $\langle i, j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

$\Downarrow$ *NamePremise* P_1

$\langle P_1 : CR(A,8) \rangle \vdash CR(F,4)$

$\Downarrow$ *Invert* $CR(F,4)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5) \lor CR(E,6) \lor CR(G,6) \lor CR(H,5)$
$\lor CR(H3) \lor CR(G2) \lor CR(E,2) \lor CR(D3)$

$\Downarrow$ *PickDisjunct* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5)$

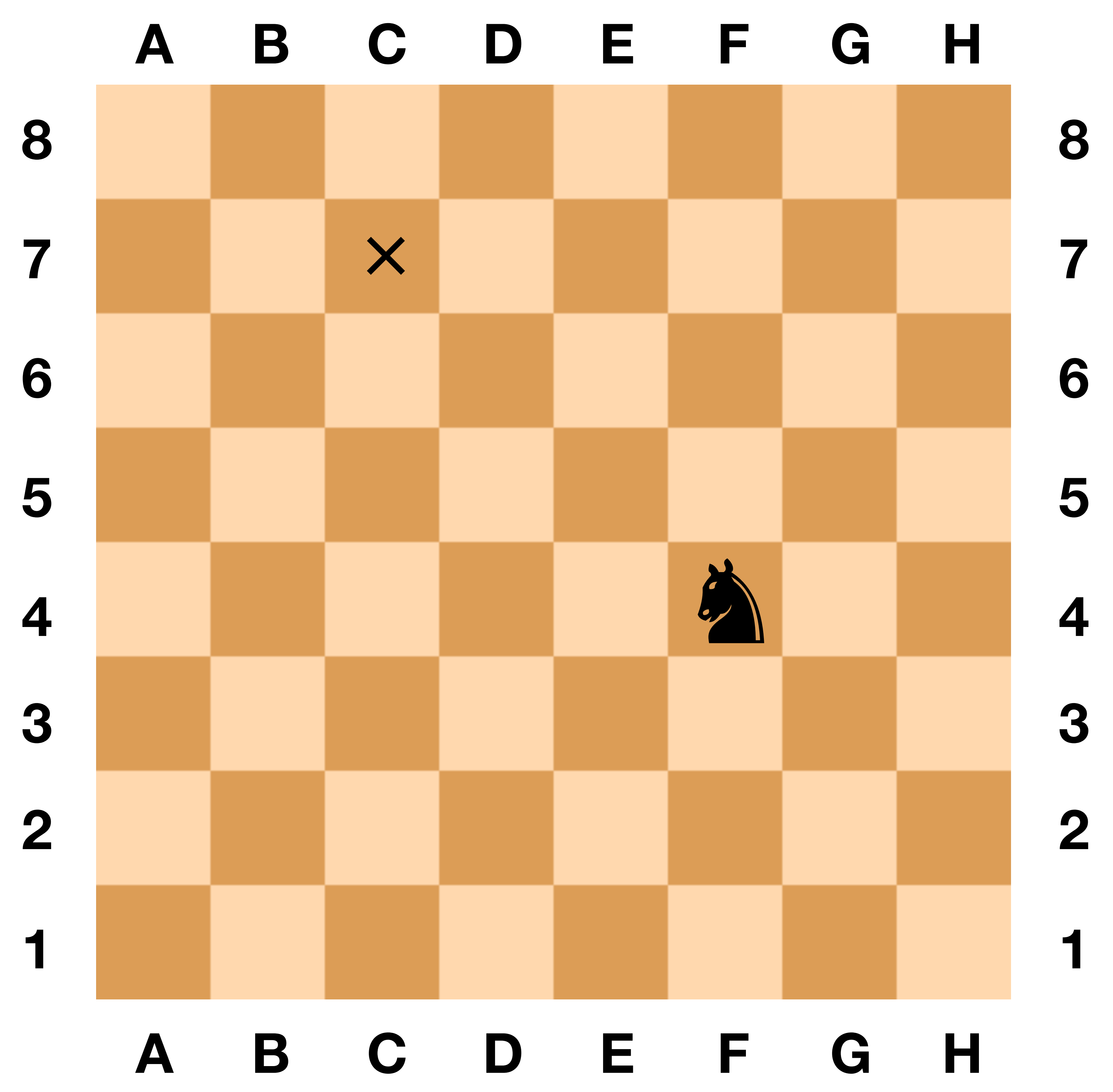
$\Downarrow$ *Invert* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7) \lor CR(B,6) \lor \ldots$

$\Downarrow$ *PickDisjunct* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7)$

What is a proof?



Definition: $CR(i, j)$: Knight CanReach the square $\langle i, j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

$\Downarrow$ *NamePremise* P_1

$\langle P_1 : CR(A,8) \rangle \vdash CR(F,4)$

$\Downarrow$ *Invert* $CR(F,4)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5) \lor CR(E,6) \lor CR(G,6) \lor CR(H,5)$
$\lor CR(H3) \lor CR(G2) \lor CR(E,2) \lor CR(D3)$

$\Downarrow$ *PickDisjunct* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5)$

$\Downarrow$ *Invert* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7) \lor CR(B,6) \lor \ldots$

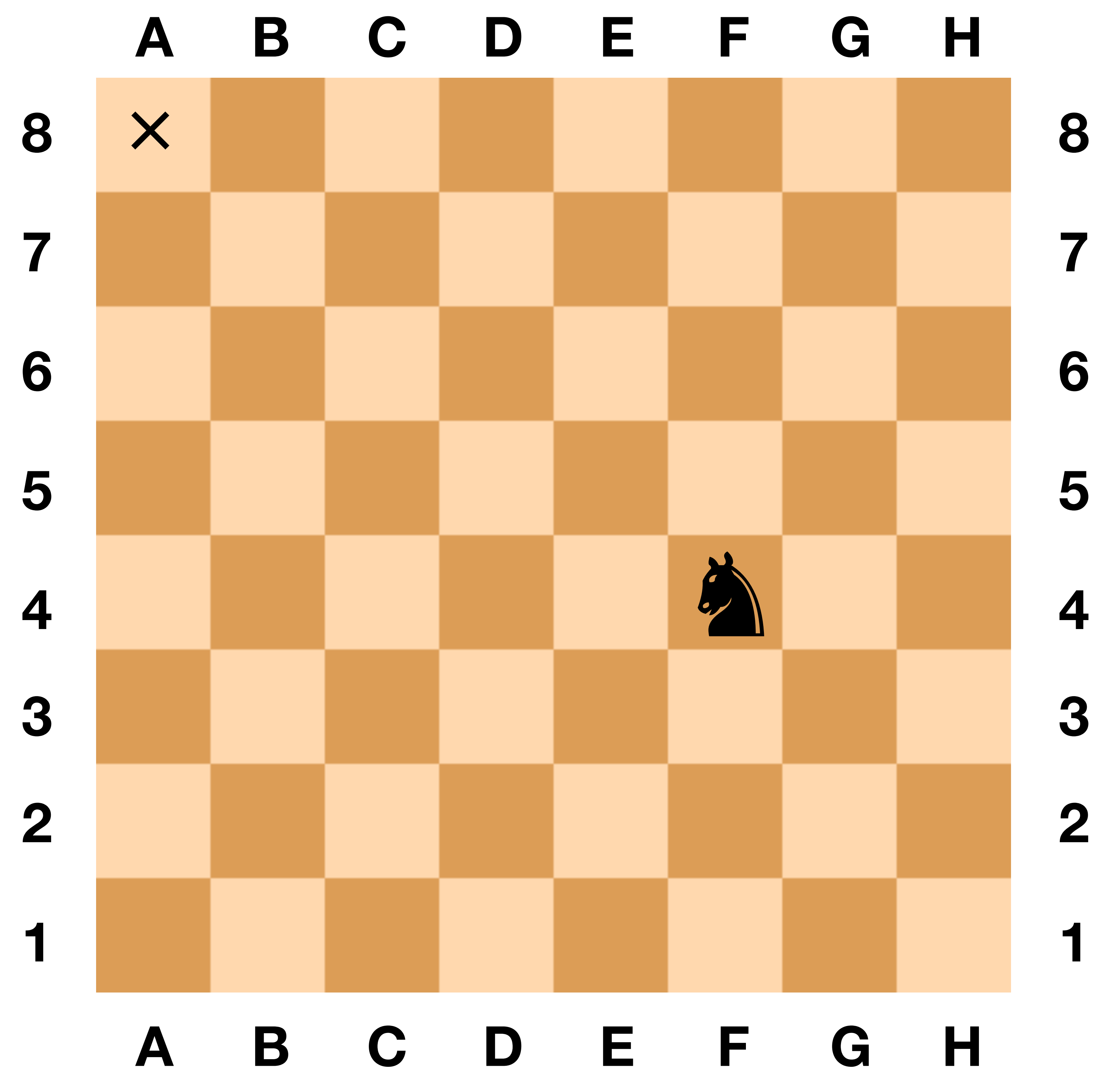
$\Downarrow$ *PickDisjunct* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7)$

$\Downarrow$ *Invert* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(A,8) \lor CR(A,6) \lor CR(E.8) \lor CR(E,6)$

What is a proof?



<u>Definition:</u> $CR(i, j)$: Knight CanReach the square $\langle i, j \rangle$

<u>Theorem</u>: $CR(A,8) \Rightarrow CR(F,4)$

$\Downarrow$ *NamePremise* P_1

$\langle P_1 : CR(A,8) \rangle \vdash CR(F,4)$

$\Downarrow$ *Invert* $CR(F,4)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5) \vee CR(E,6) \vee CR(G,6) \vee CR(H,5)$
$\vee CR(H3) \vee CR(G2) \vee CR(E,2) \vee CR(D3)$

$\Downarrow$ *PickDisjunct* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5)$

$\Downarrow$ *Invert* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7) \vee CR(B,6) \vee \ldots$

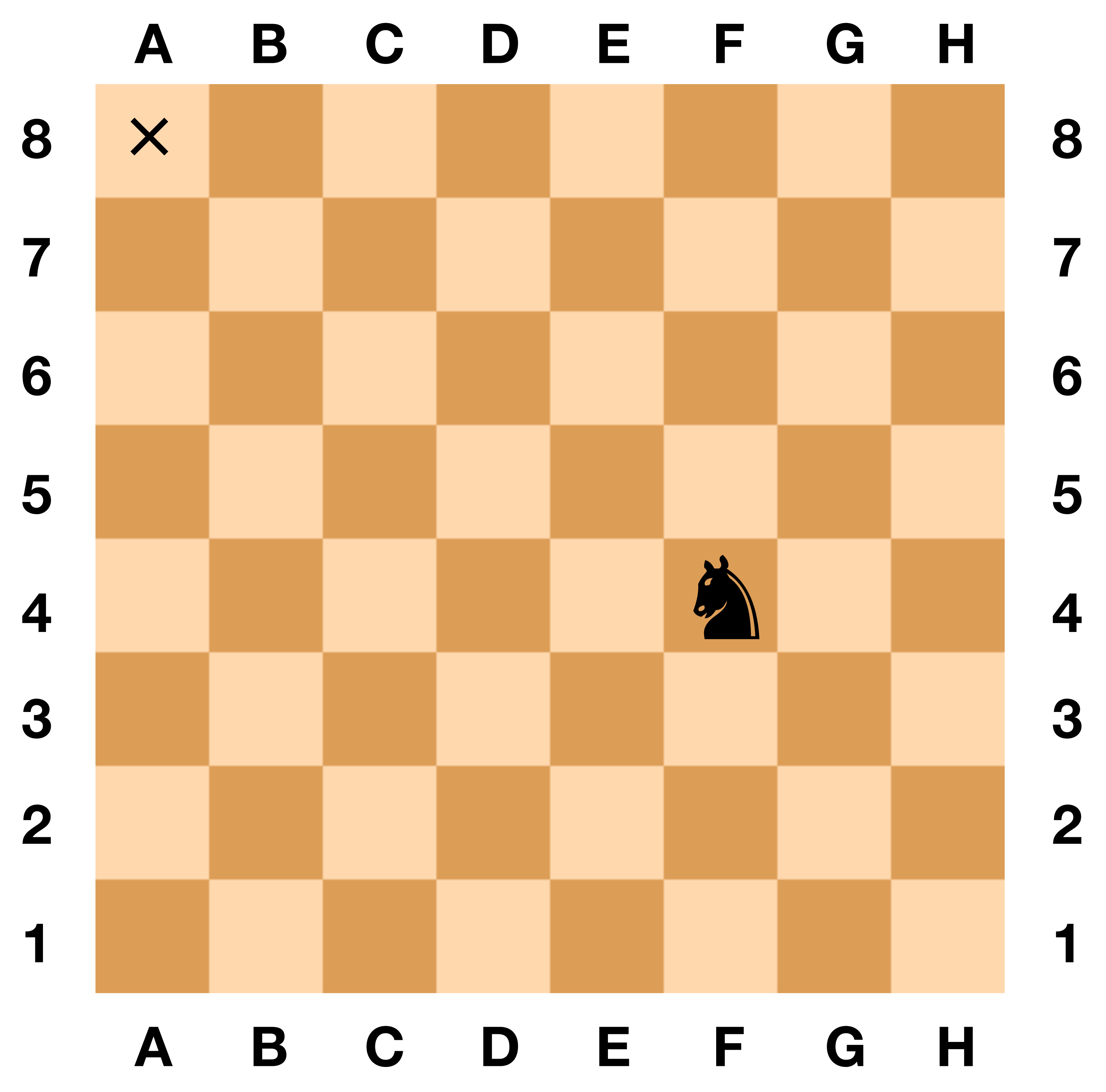
$\Downarrow$ *PickDisjunct* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7)$

$\Downarrow$ *Invert* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(A,8) \vee CR(A,6) \vee CR(E.8) \vee CR(E,6)$ $\Rightarrow$ *PickDisjunct* $CR(A,8)$ $\langle P_1 : CR(A,8) \rangle \vdash CR(A,8)$

What is a proof?



<u>Definition:</u> $CR(i,j)$: Knight CanReach the square $\langle i,j \rangle$

<u>Theorem</u>: $CR(A,8) \Rightarrow CR(F,4)$

$\Downarrow$ *NamePremise* P_1

$\langle P_1 : CR(A,8) \rangle \vdash CR(F,4)$

$\Downarrow$ *Invert* $CR(F,4)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5) \vee CR(E,6) \vee CR(G,6) \vee CR(H,5)$
$\vee\ CR(H3) \vee CR(G2) \vee CR(E,2) \vee CR(D3)$

$\Downarrow$ *PickDisjunct* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(D,5)$

$\Downarrow$ *Invert* $CR(D,5)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7) \vee CR(B,6) \vee \ldots$

$\Downarrow$ *PickDisjunct* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(C,7)$

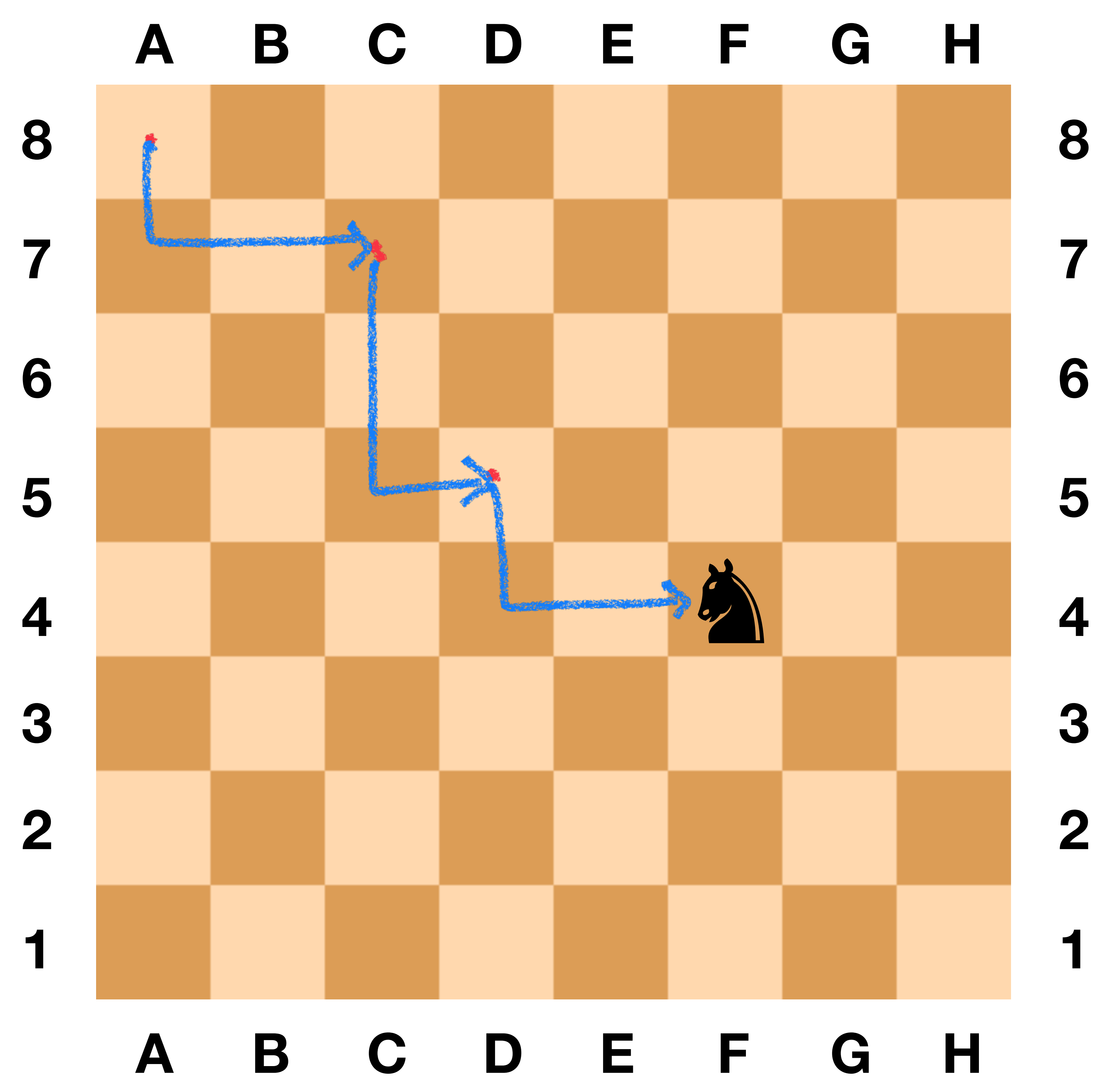
$\Downarrow$ *Invert* $CR(C,7)$

$\langle P_1 : CR(A,8) \rangle \vdash CR(A,8) \vee CR(A,6) \vee CR(E.8) \vee CR(E,6)$ $\Rightarrow$ *PickDisjunct* $CR(A,8)$ $\langle P_1 : CR(A,8) \rangle \vdash CR(A,8)$

$\Uparrow$ *ApplyPremise* P_1

true

What is a proof?



Definition: $CR(i, j)$: Knight CanReach the square $\langle i, j \rangle$

Theorem: $CR(A,8) \Rightarrow CR(F,4)$

$NamePremise\ P_1$
$Invert\ CR(F,4)$
$PickDisjunct\ CR(D,5)$
$Invert\ CR(D,5)$
$PickDisjunct\ CR(C,7)$
$Invert\ CR(C,7)$
$PickDisjunct\ CR(A,8)$
$ApplyPremise\ P_1$

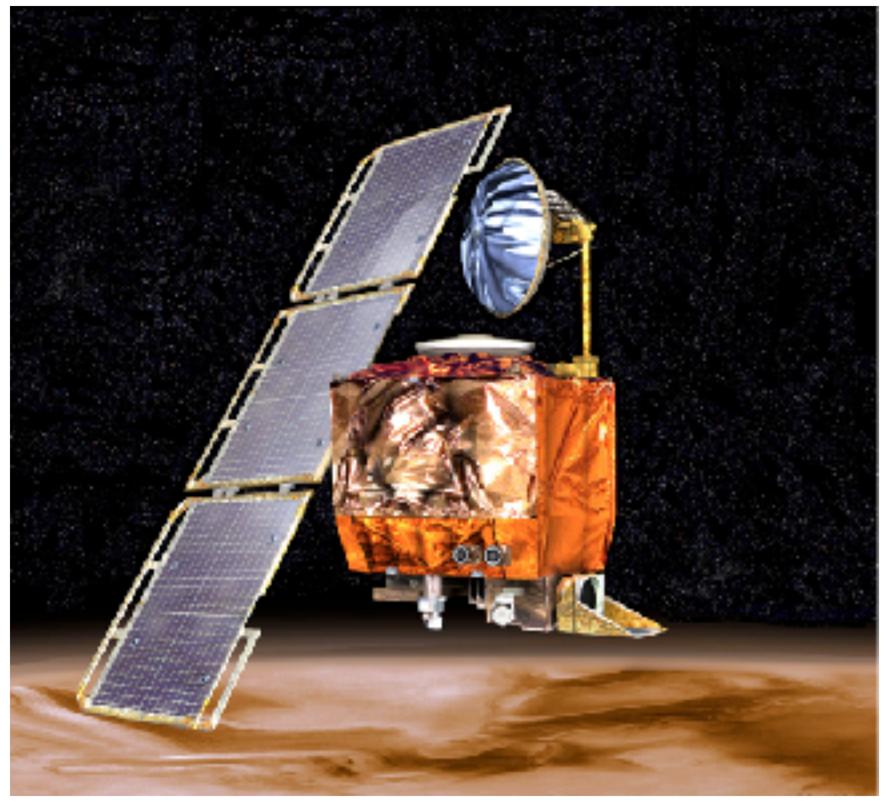
} A machine-checkable *proof script*!

We are going to apply such rigorous standards to build proofs of program correctness.

Why verify programs?

Because building reliable software is hard. *Really* hard!

Therac 25



Mars Climate Orbiter

Boeing 737 Max 8

"*Program testing can be used to show the presence of bugs, but never to show their absence!*" - E W Dijkstra

Does Program Verification Scale?

Use of formal methods to verify full-scale software systems is a hot research topic!

- CompCert – fully verified C compiler
  Leroy, INRIA
- Vellvm – formalized LLVM IR
  Zdancewic, Penn
- Ynot – verified DBMS, web services
  Morrisett, Harvard
- Verified Software Toolchain
  Appel, Princeton
- Bedrock – web programming, packet filters
  Chlipala, MIT
- CertiKOS – certified OS kernel
  Shao & Ford, Yale

[Slide courtesy: B C Pierce]

Does Program Verification Pay?

SiFive

diem

amazon

galois

BAE SYSTEMS

Tezos

facebook

NASA

Coq

Thierry Coquand

Invented Calculus of Inductive Constructions — theoretical basis for Coq

- A mechanized *proof assistant*.
  - Checks if the proof you write indeed proves the theorem you state.
- We make extensive use of Coq in this class.
- Installing Coq (version 8.12 or later):
  - From https://coq.inria.fr: You can download a Coq platform binary that includes a dedicated IDE for Coq called Coqide.
  - From https://proofgeneral.github.io: Installs a Coq major mode for Emacs. Best option if you are already familiar with Emacs.
  - Via Opam — the package manager of OCaml. See https://coq.inria.fr/opam-using.html for instructions. You can combine this with vscoq plugin for vscode: https://github.com/coq-community/vscoq.

Evaluation Components

Item	Count	Cumulative Weight
Homeworks	8 of 10	40%
Mid-term	1	20%
Course project based on a research paper	1	15%
Final	1	25%

- Coq Assignments: Write proofs in Coq for select exercises.
- One homework each week for 10 weeks. Best 8 scores count towards final grade.
- Due each Friday before the class. Upload your submissions on Canvas (link will be posted on course website).
- Collaboration is permitted. Plagiarism is not!

Evaluation Components

Item	Count	Cumulative Weight
Homeworks	8 of 10	40%
Mid-term	1	20%
Course project based on a research paper	1	15%
Final	1	25%

- Written exams.
- Mid-term will be in the class. Sometime in October. Date TBD.
- Final in December. Date and place TBD.
- Doing homework assignments and textbook exercises is a good practice for exams.

Evaluation Components

Item	Count	Cumulative Weight
Homeworks	8 of 10	40%
Mid-term	1	20%
Course project based on a research paper	1	15%
Final	1	25%

- Select a research paper from PLDI/ POPL/ OSDI/ SOSP/ NSDI/ SIGCOMM / NeurIPS/ CVPR; formalize and prove their meta-theory in Coq.
- Alternatively: Formalize a model of a real-world system, and prove interesting properties.
  - Eg: Border Gateway Protocol (BGP) guarantees absence of routing loops.
- Can be done alone or in groups of two. Expectations are scaled accordingly.
- **Important:** Talk to me before you start the project!

TODO for you

- Checkout course website: https://csci5535.github.io
- Install Coq (v8.12 or later).
- Register on course Piazza (link on course website).
- Read Preface and Basics chapters from textbook Vol 1 (Logical Foundations)
- Download and run (step through) Basics.v file in your chosen Coq IDE.