

First-Order Logic = FOL = $\mathcal{L}_1$, Part 3

Selmer Bringsjord

Rensselaer AI & Reasoning (RAIR) Lab
Department of Cognitive Science
Department of Computer Science
Lally School of Management & Technology
Rensselaer Polytechnic Institute (RPI)
Troy, New York 12180 USA

Intro to Logic-based AI
10/8/2026



Pedagogy Here

Pedagogy Here

Learning Trajectory From HS;
Travel It, Learn It; Grade of A

Pedagogy Here

Learning Trajectory From HS;
Travel It, Learn It; Grade of A

Glimpses of Logic-Based
AI in “The Show”

Logic-&-AI In The News (2026): A Very Important Paper

arXiv:2610.08144v1 [math.AP] 6 Oct 2026

NAVIER-STOKES LOST IN TRANSLATION

WHY LEAN VERIFICATION OF AI AUTOFORMALISATION DOES NOT GUARANTEE CORRECT NATURAL LANGUAGE PROOFS

A. BASTOUNIS, F. CIRCELLI, AND A. C. HANSEN*

ABSTRACT. Autoformalisation is increasingly used to verify mathematical texts, including those generated by AI, as in OpenAI's announced proof of blow-up of solutions to the Navier-Stokes equations. In this process, an AI system translates the text from a natural language (NL) into a formal language such as Lean. Once this translation is done, the argument expressed in the formal language can easily be mechanically verified. The purpose of this article is to demonstrate why this process may offer no confidence in the original NL argument, owing to the various difficulties in performing the translation semantically faithfully. In particular, we highlight that the problem of resolving ambiguities in mathematical NL text, which is necessary in order to provide semantically faithful translation, is arbitrarily high up in the Solvability Complexity Index (SCI) hierarchy/arithmetic hierarchy (the $SCI = \infty$). Hence, informally, providing semantically faithful AI autoformalisation is harder than any computational problem including the Halting problem (which has $SCI = 1$). To demonstrate the effect of this result we provide several examples of AI mistranslations of NL statements and proofs into Lean in practice, resulting in mismatches between NL proofs and their Lean 'verifications'. These include OpenAI's announced Navier-Stokes proof. In particular, we show that the formalised Lean proof does not correspond to the NL proof of blow-up of solutions to the Navier-Stokes equations.

1. INTRODUCTION: TWO TYPES OF AUTOFORMALISATION

We start by describing the two main types of autoformalisation from a natural language (NL) to a formal language:

- (i) **Translating an alleged proof in an NL to obtain a formal Lean proof.** Suppose a theorem (the statement, not including a proof) has already been faithfully and correctly translated into Lean. Suppose also the theorem is accompanied by an alleged NL proof. The goal is to let a machine (which we will call AI) translate the NL proof into Lean to obtain a formal proof of the theorem. Success is declared when the AI has produced a 'translation' which Lean accepts as a valid proof (meaning it compiles), with no use of Lean's 'sorry' keyword or extra axioms being permitted.
- (ii) **Faithful semantic translation of a mathematical text.** Suppose we are given a mathematical text (e.g. a book or a paper) written in an NL. The goal is to let an AI translate every definition, theorem, proposition, proof etc. in the text to Lean code that compiles, see e.g. the example in §5.1. The definitions, statements and proofs must themselves be translated, semantically preserving the source's mathematical content. For a book, this obligation applies throughout and includes the dependencies between concepts and results. If the stated goal is to validate its proofs, correspondence to the written arguments must also be established. In particular, a different, provable assertion would not constitute a semantically faithful translation. This autoformalisation problem is much harder than problem (i) above (see §4).

Warning: Meeting the success criterion of (i) does not mean that the original NL proof is correct. Indeed, a full Lean compilation of the 'translation' does not guarantee a semantically faithful translation. The AI could very easily change the semantics of the translation in order to satisfy this success criterion. For example, the

2020 *Mathematics Subject Classification.* 35Q30, 03Dxx (primary) and 68V20, 68Txx, 03B65 (secondary).

Key words and phrases. AI Autoformalisation, Navier-Stokes equations, Solvability Complexity Index hierarchy, boundaries of AI, Hilbert's 10th problem.

*Corresponding author: a.hansen@damtp.cam.ac.uk.

<https://arxiv.org/abs/2610.08144>

Back to Learning FOL

...

Our Final New Inference Rule in FOL

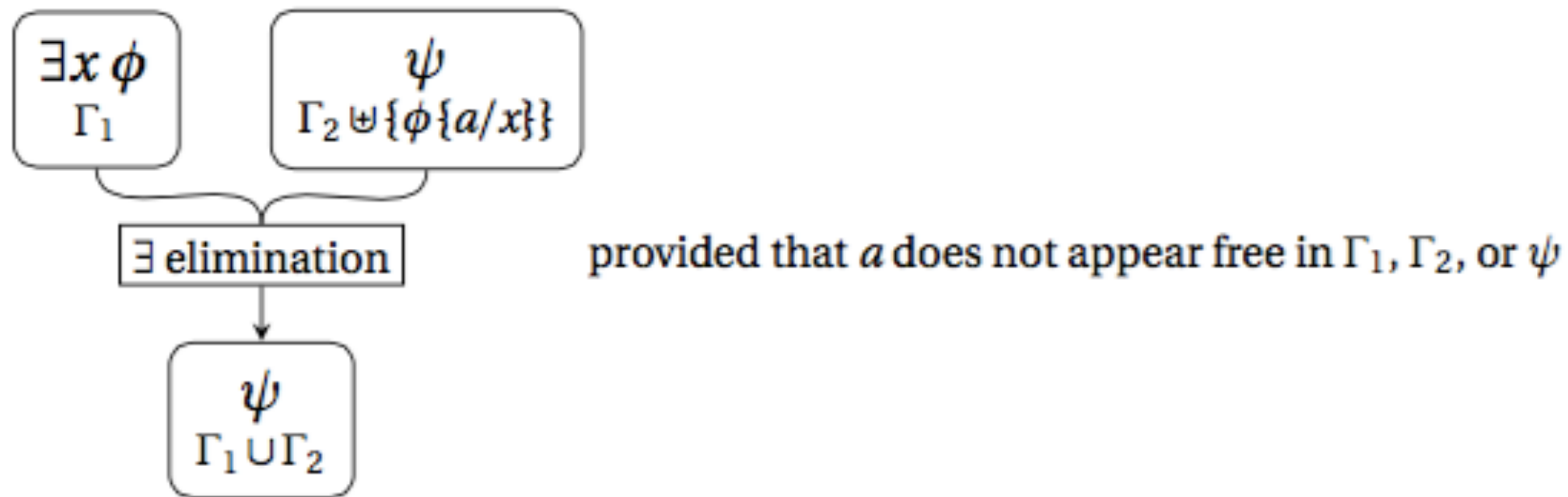
Our Final New Inference Rule in FOL

- existential elimination (intuitively put):

Our Final New Inference Rule in FOL

- existential elimination (intuitively put):
 - If we know that (i) there's something x which is an R , and (ii) on the supposition that a is an arbitrary representative (a “witness”) of such an x we can prove P , then we are permitted to deduce P from (i) alone.

existential elimination, precise version:

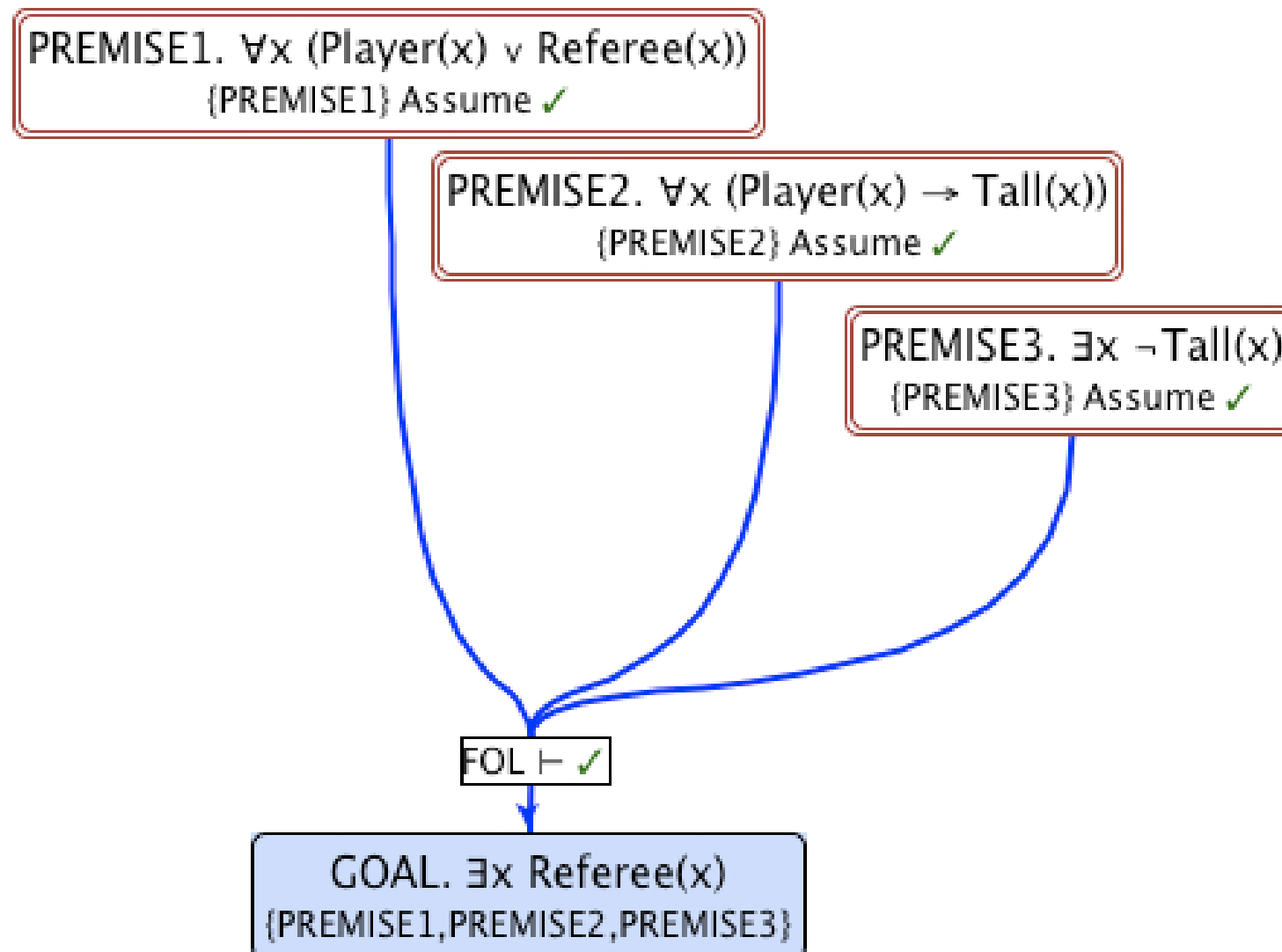


Starting State of Workspace

(Assumes a domain of e.g. players on a
March-madness basketball court.)

Starting State of Workspace

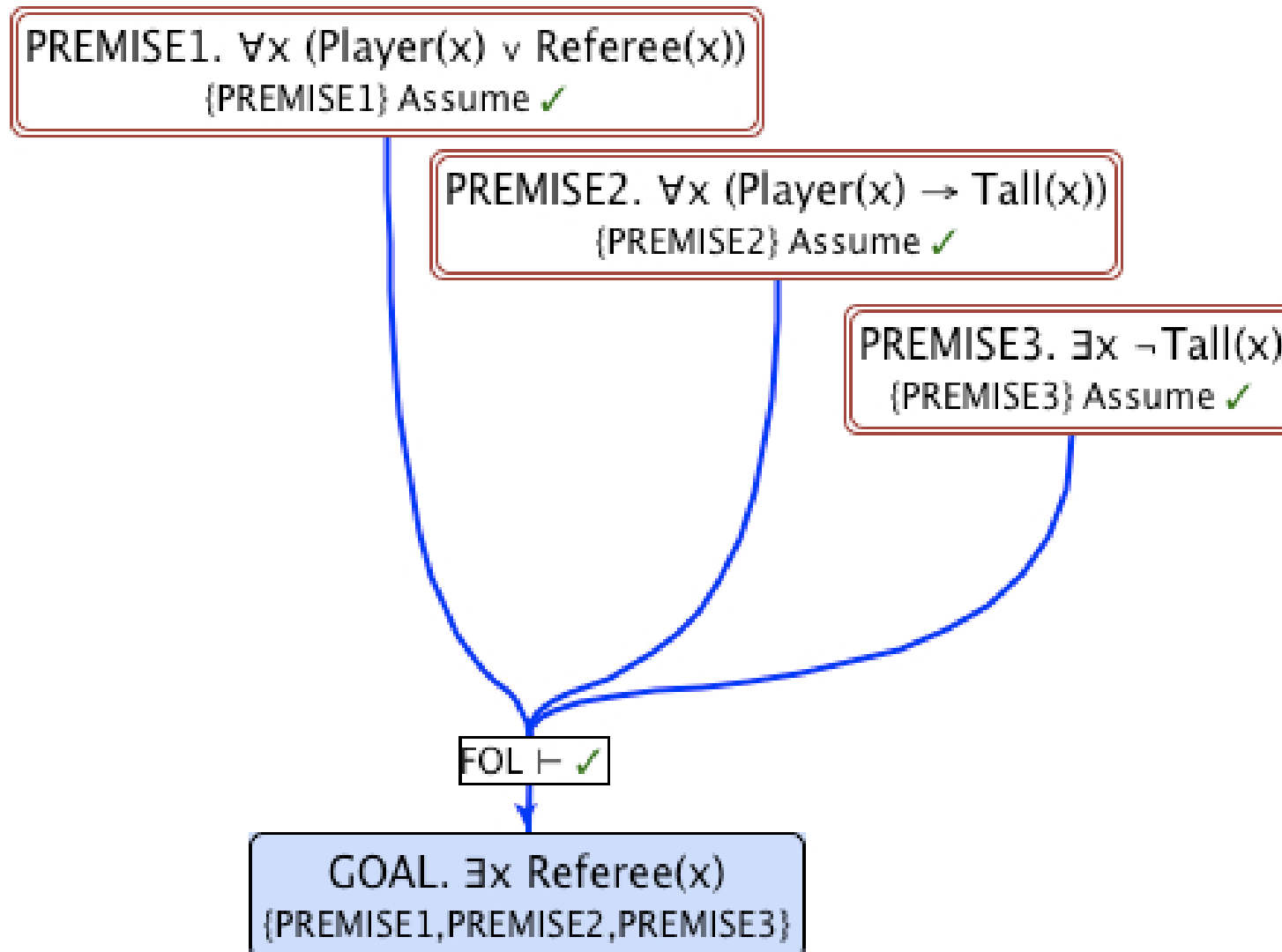
(Assumes a domain of e.g. players on a March-madness basketball court.)



Starting State of Workspace

(Assumes a domain of e.g. players on a March-madness basketball court.)

“Each and every thing is
either a player or a referee.”



Starting State of Workspace

(Assumes a domain of e.g. players on a March-madness basketball court.)

“Each and every thing is
either a player or a referee.”

PREMISE1. $\forall x (\text{Player}(x) \vee \text{Referee}(x))$
{PREMISE1} Assume ✓

“All players are tall.”

PREMISE2. $\forall x (\text{Player}(x) \rightarrow \text{Tall}(x))$
{PREMISE2} Assume ✓

PREMISE3. $\exists x \neg \text{Tall}(x)$
{PREMISE3} Assume ✓

FOL $\vdash$ ✓

GOAL. $\exists x \text{Referee}(x)$
{PREMISE1, PREMISE2, PREMISE3}

Starting State of Workspace

(Assumes a domain of e.g. players on a March-madness basketball court.)

“Each and every thing is
either a player or a referee.”

PREMISE1. $\forall x (\text{Player}(x) \vee \text{Referee}(x))$
{PREMISE1} Assume ✓

“All players are tall.”

PREMISE2. $\forall x (\text{Player}(x) \rightarrow \text{Tall}(x))$
{PREMISE2} Assume ✓

PREMISE3. $\exists x \neg \text{Tall}(x)$
{PREMISE3} Assume ✓

“There’s a non-tall thing.”

FOL $\vdash$ ✓

GOAL. $\exists x \text{Referee}(x)$
{PREMISE1, PREMISE2, PREMISE3}

Step 1

PREMISE1. $\forall x (\text{Player}(x) \vee \text{Referee}(x))$
{PREMISE1} Assume ✓

PREMISE2. $\forall x (\text{Player}(x) \rightarrow \text{Tall}(x))$
{PREMISE2} Assume ✓

PREMISE3. $\exists x \neg \text{Tall}(x)$
{PREMISE3} Assume ✓

5. $\neg \text{Tall}(a)$
{5} Assume ✓

GOAL. $\exists x \text{Referee}(x)$
FOL $\vdash$ ✗

Step I

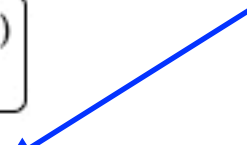
PREMISE1. $\forall x (\text{Player}(x) \vee \text{Referee}(x))$
{PREMISE1} Assume ✓

PREMISE2. $\forall x (\text{Player}(x) \rightarrow \text{Tall}(x))$
{PREMISE2} Assume ✓

PREMISE3. $\exists x \neg \text{Tall}(x)$
{PREMISE3} Assume ✓

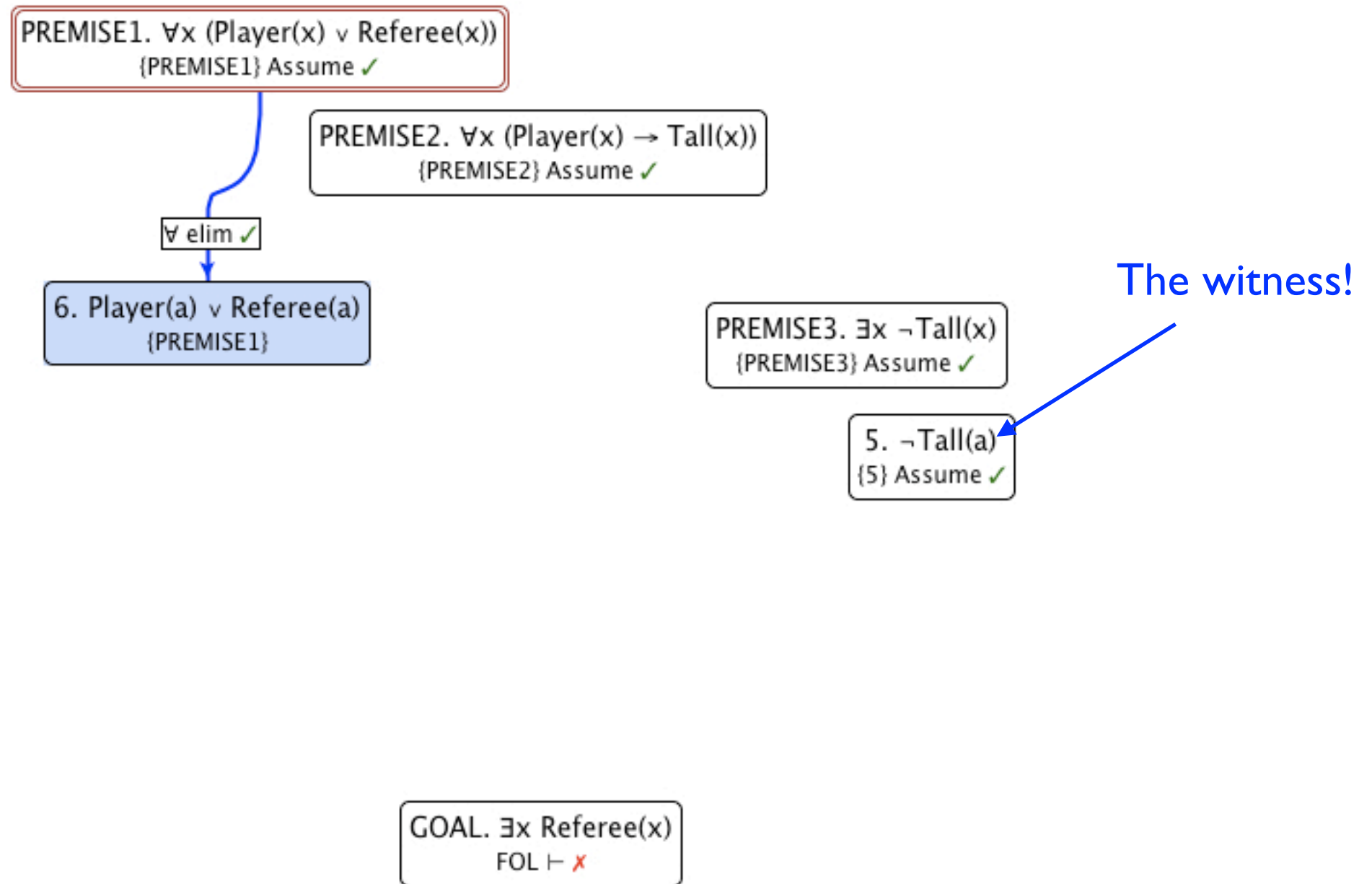
5. $\neg \text{Tall}(a)$
{5} Assume ✓

The witness!

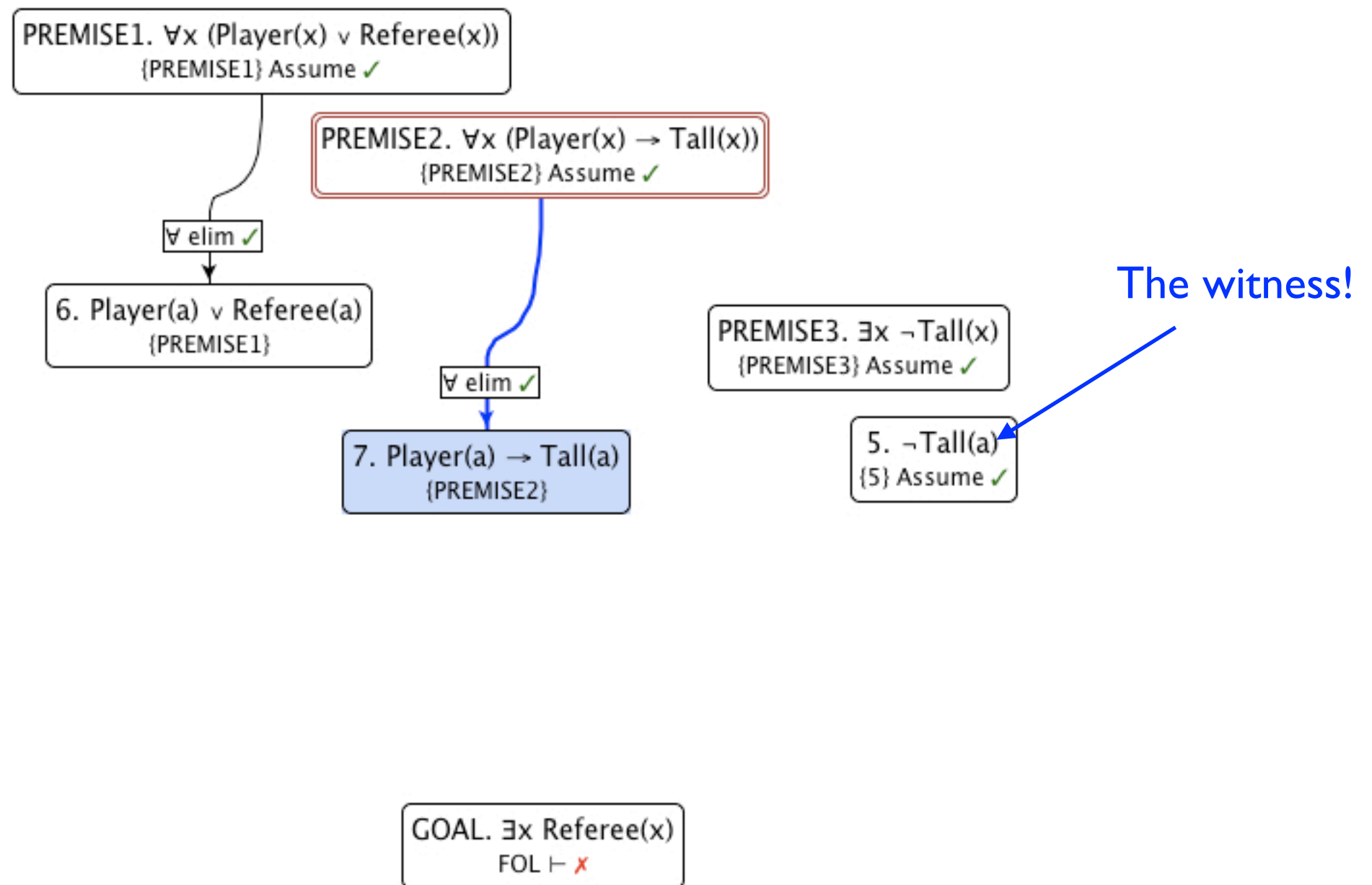


GOAL. $\exists x \text{Referee}(x)$
FOL $\vdash$ ✗

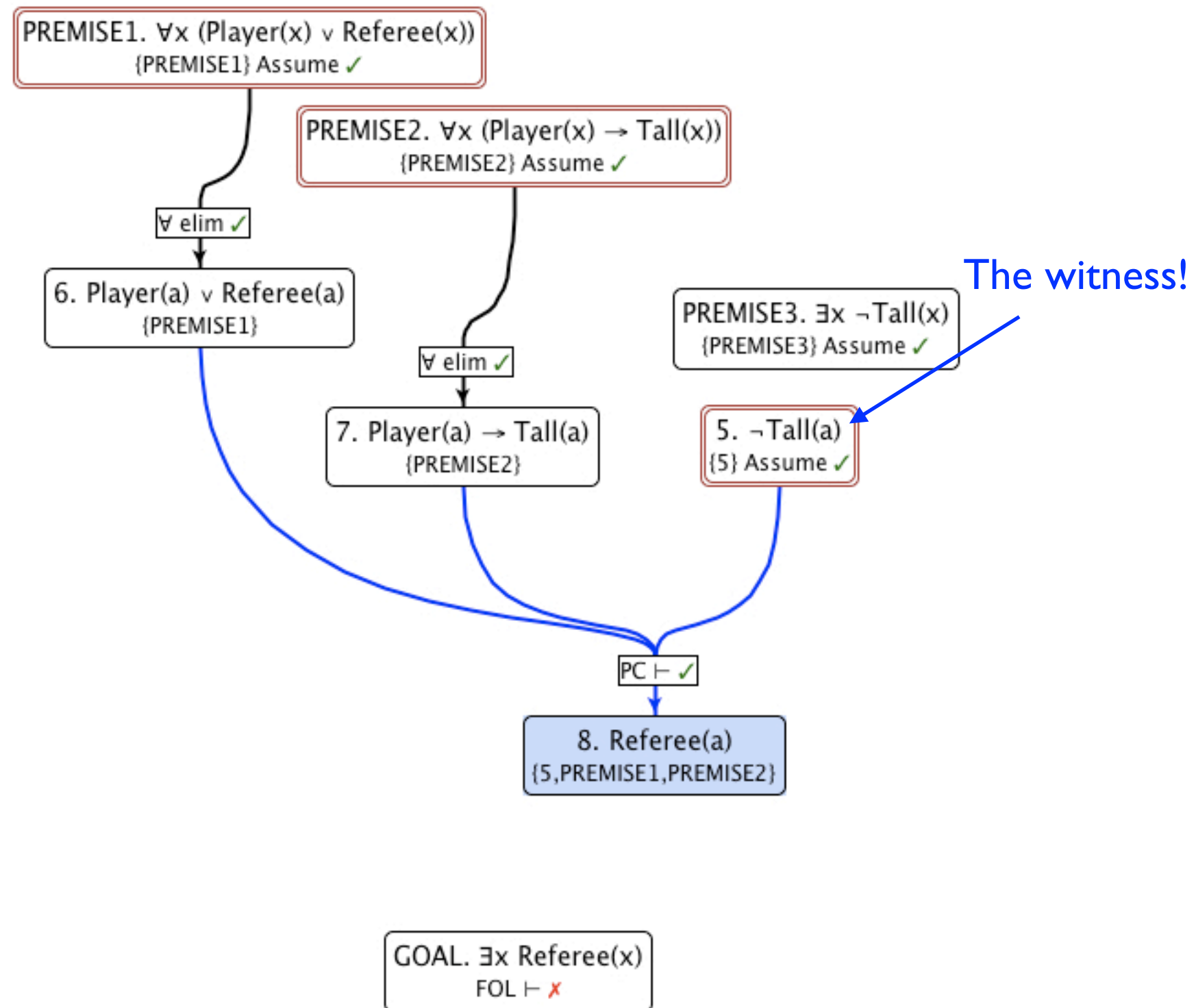
Step 2



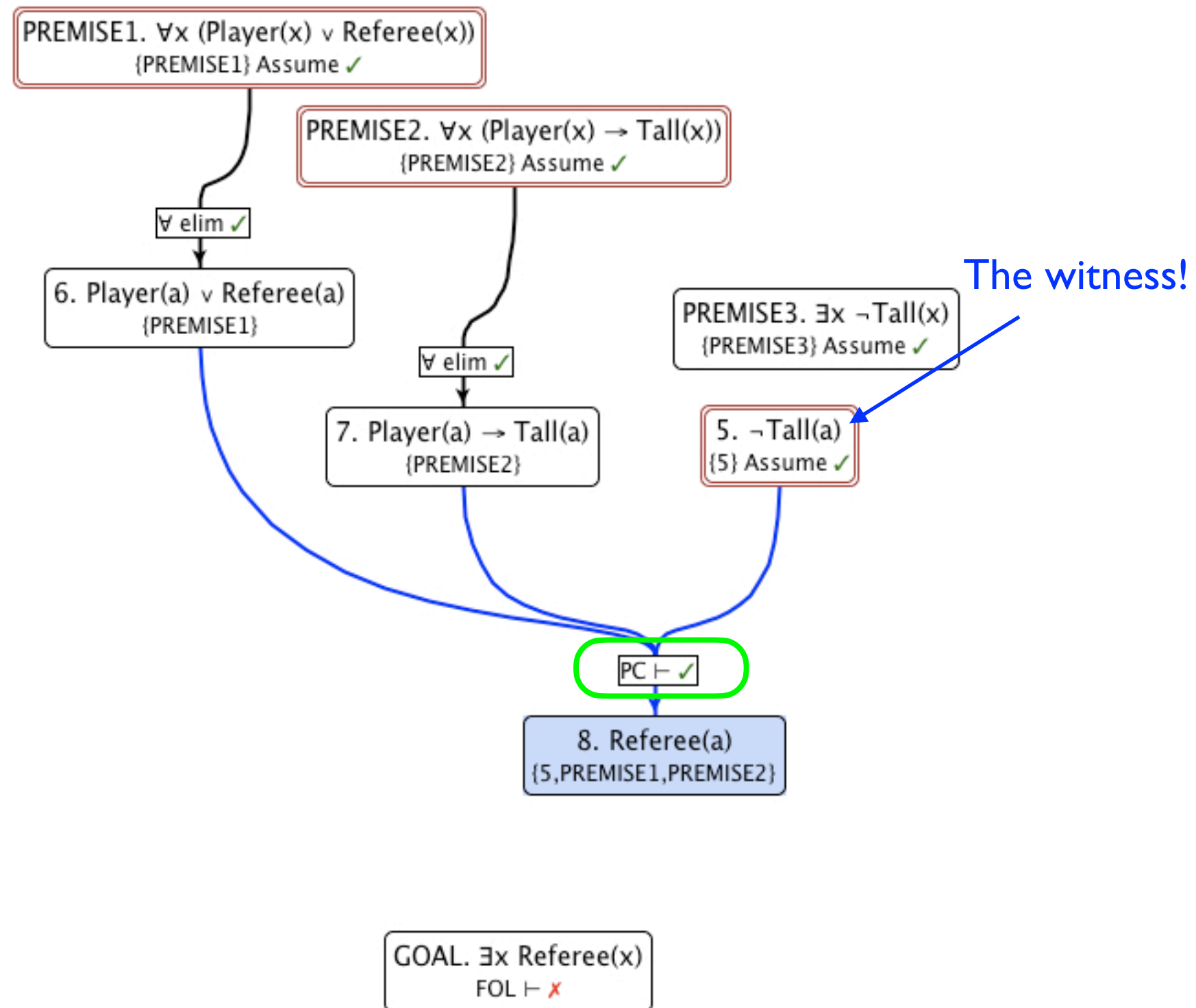
Step 3



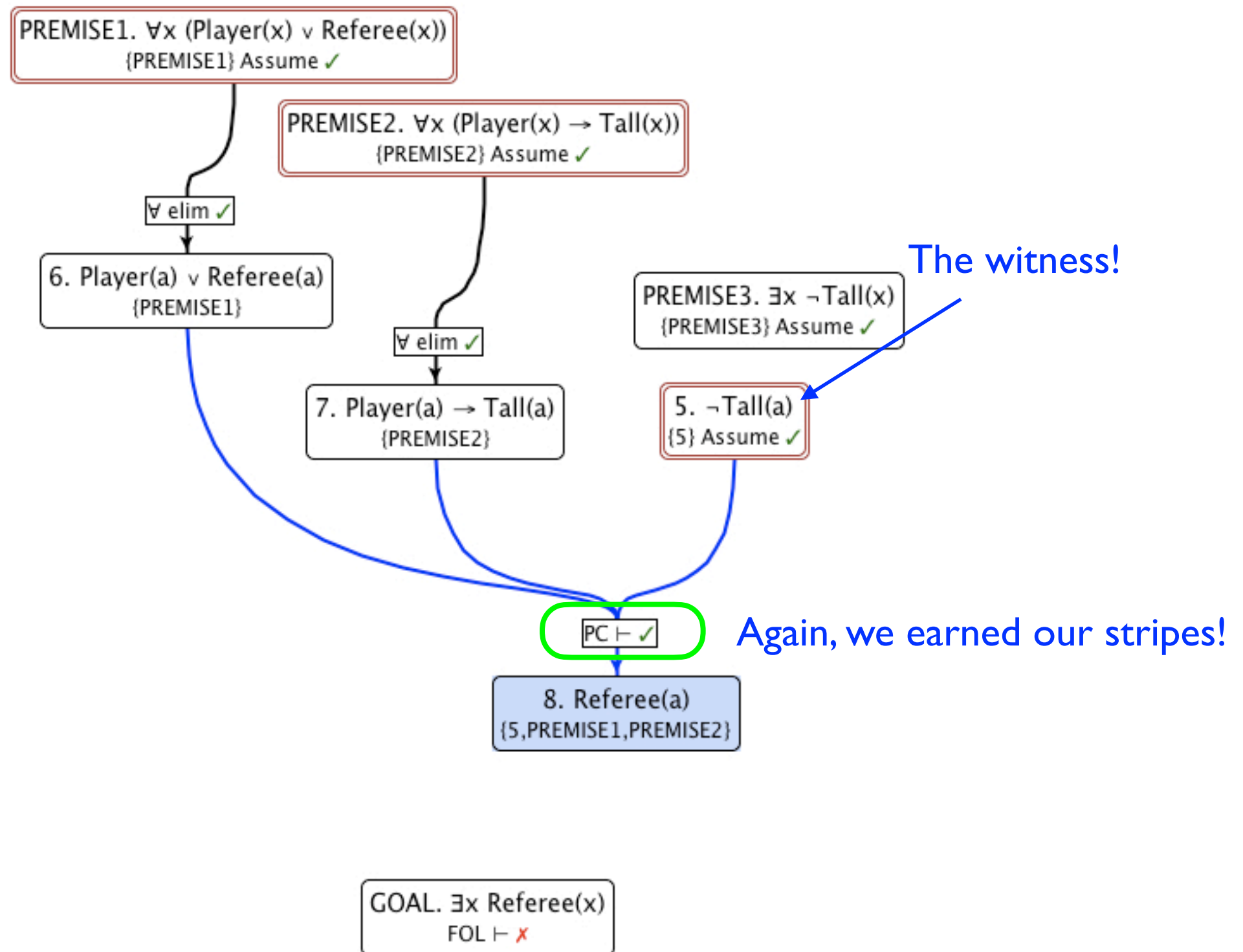
Step 4



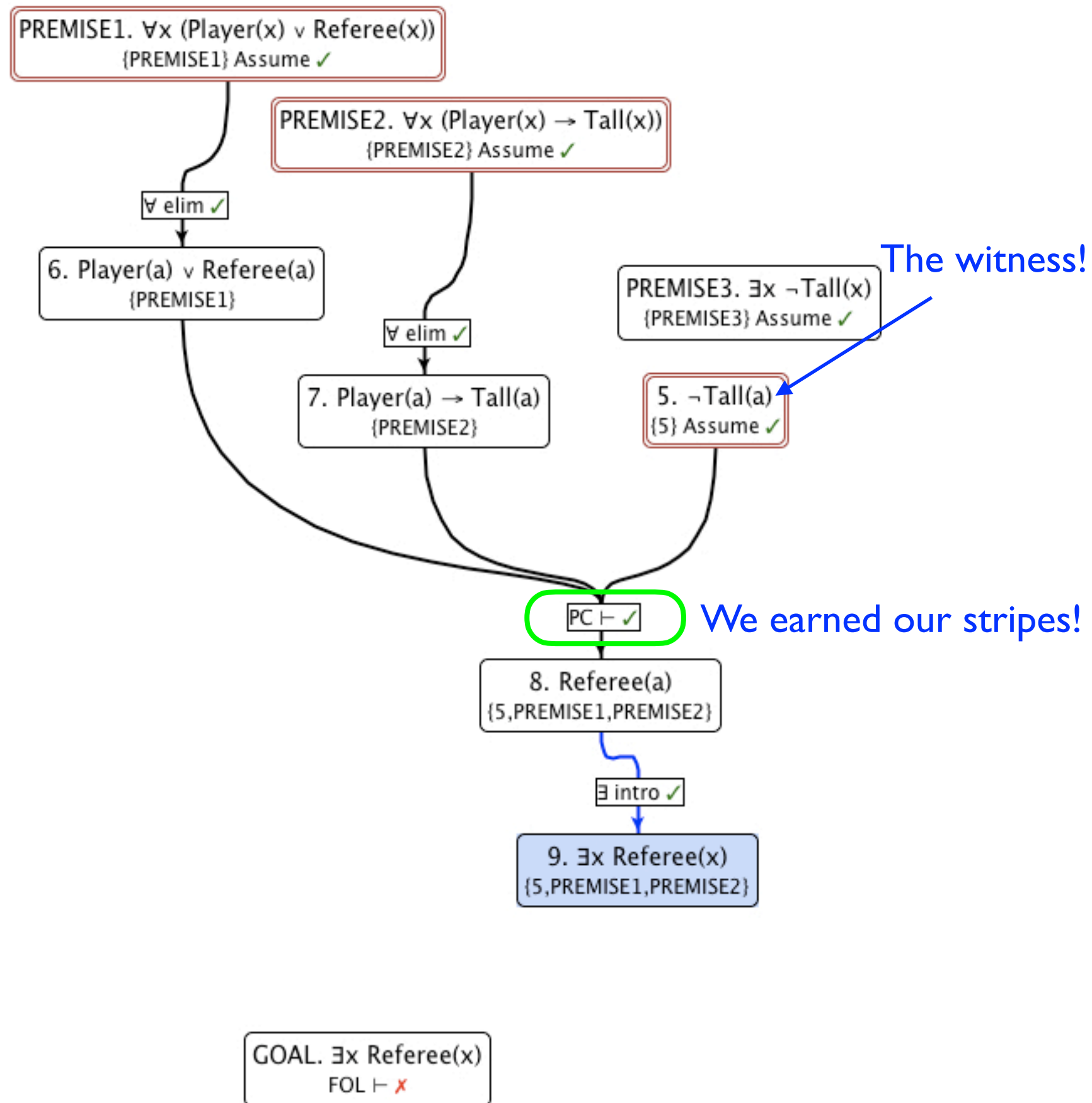
Step 4



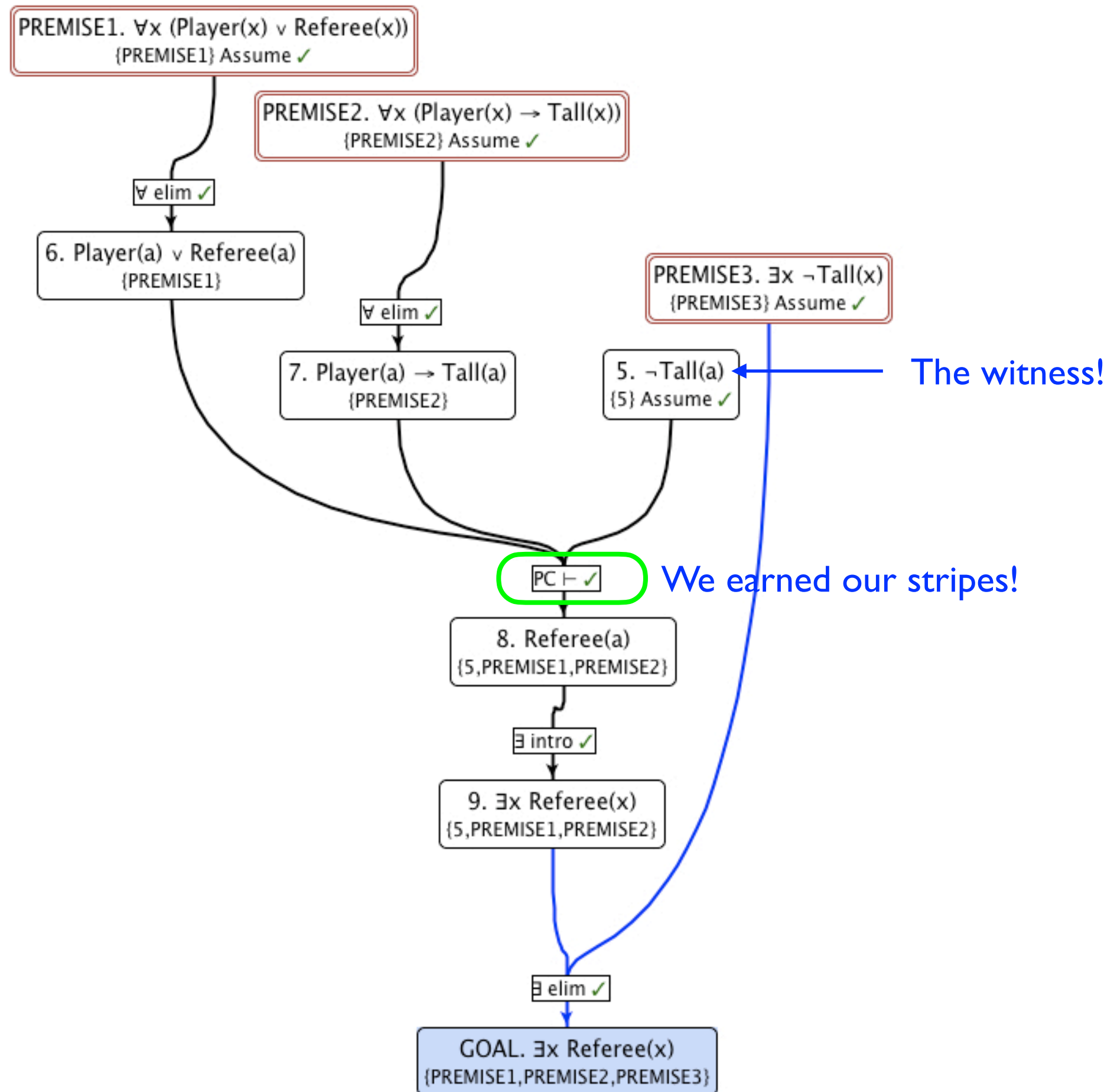
Step 4



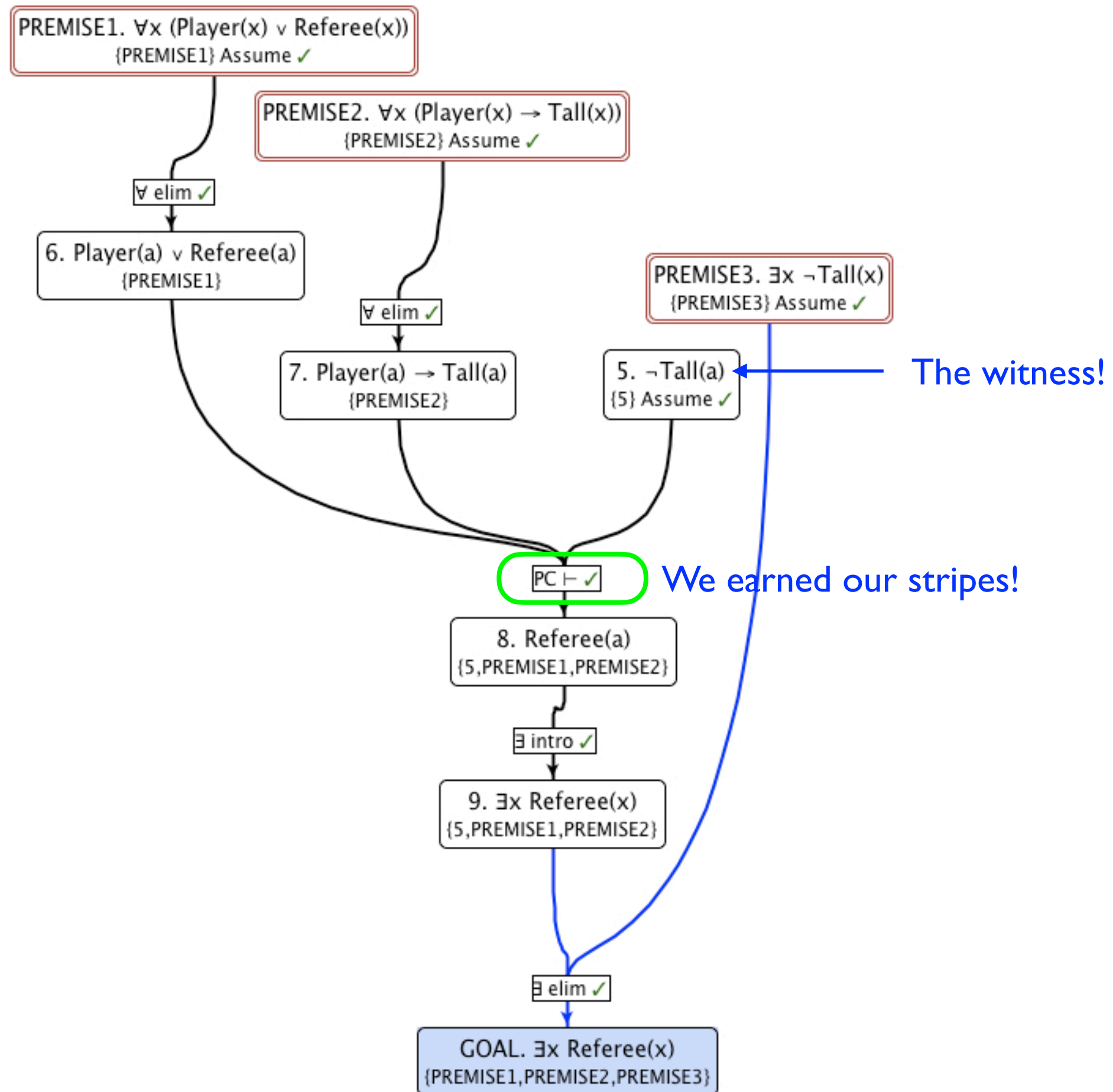
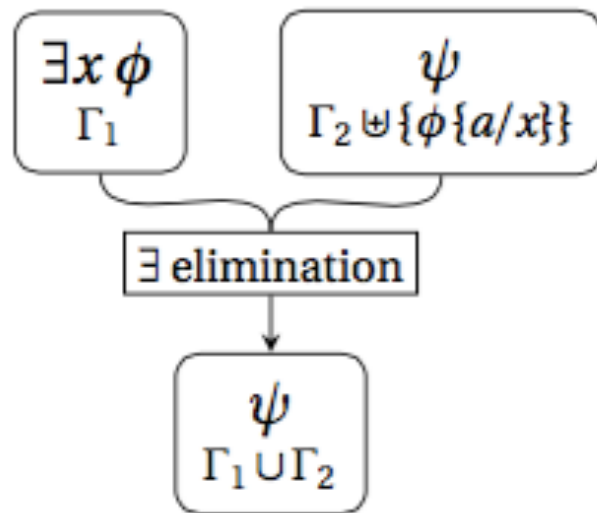
Step 5



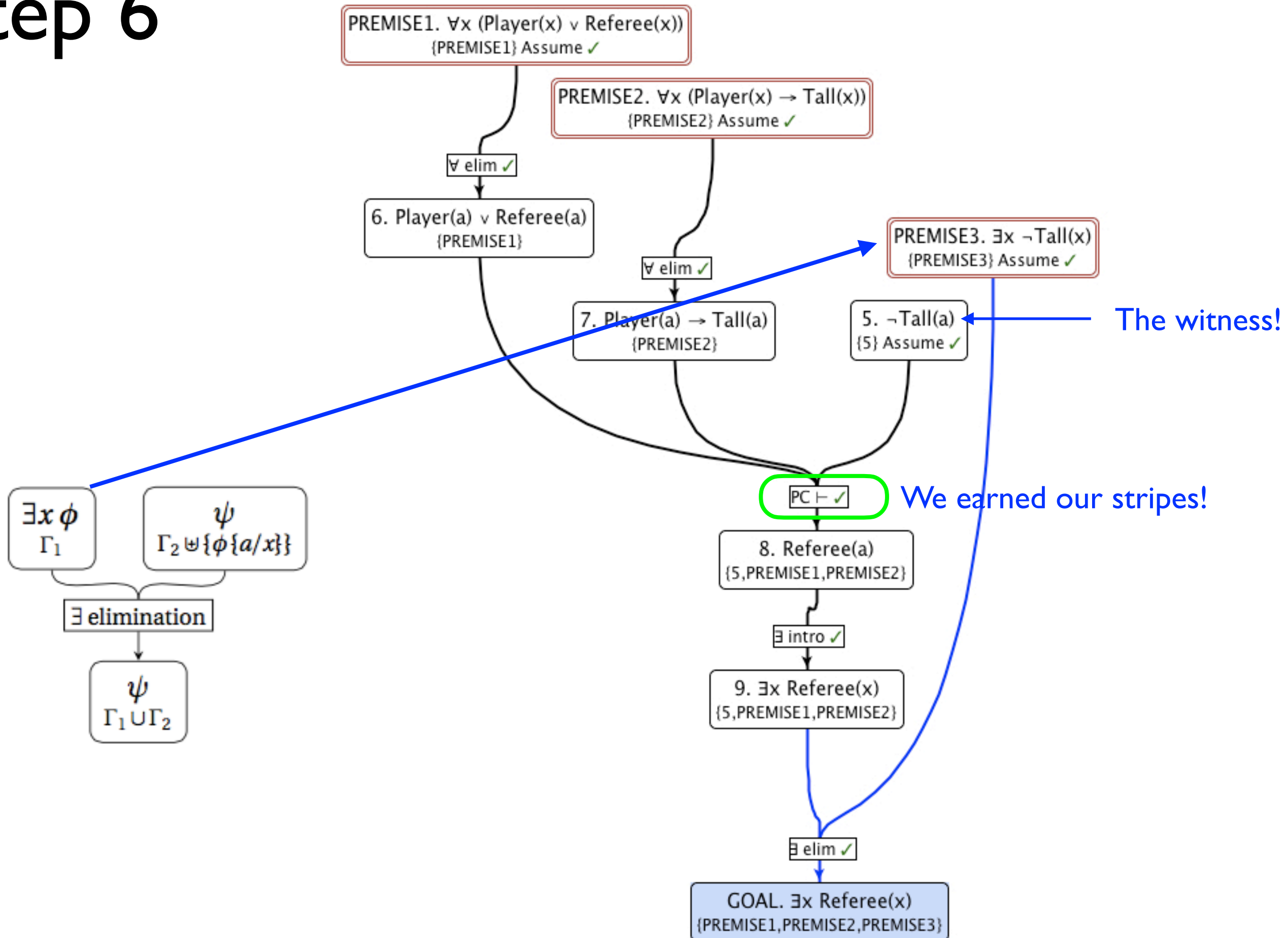
Step 6



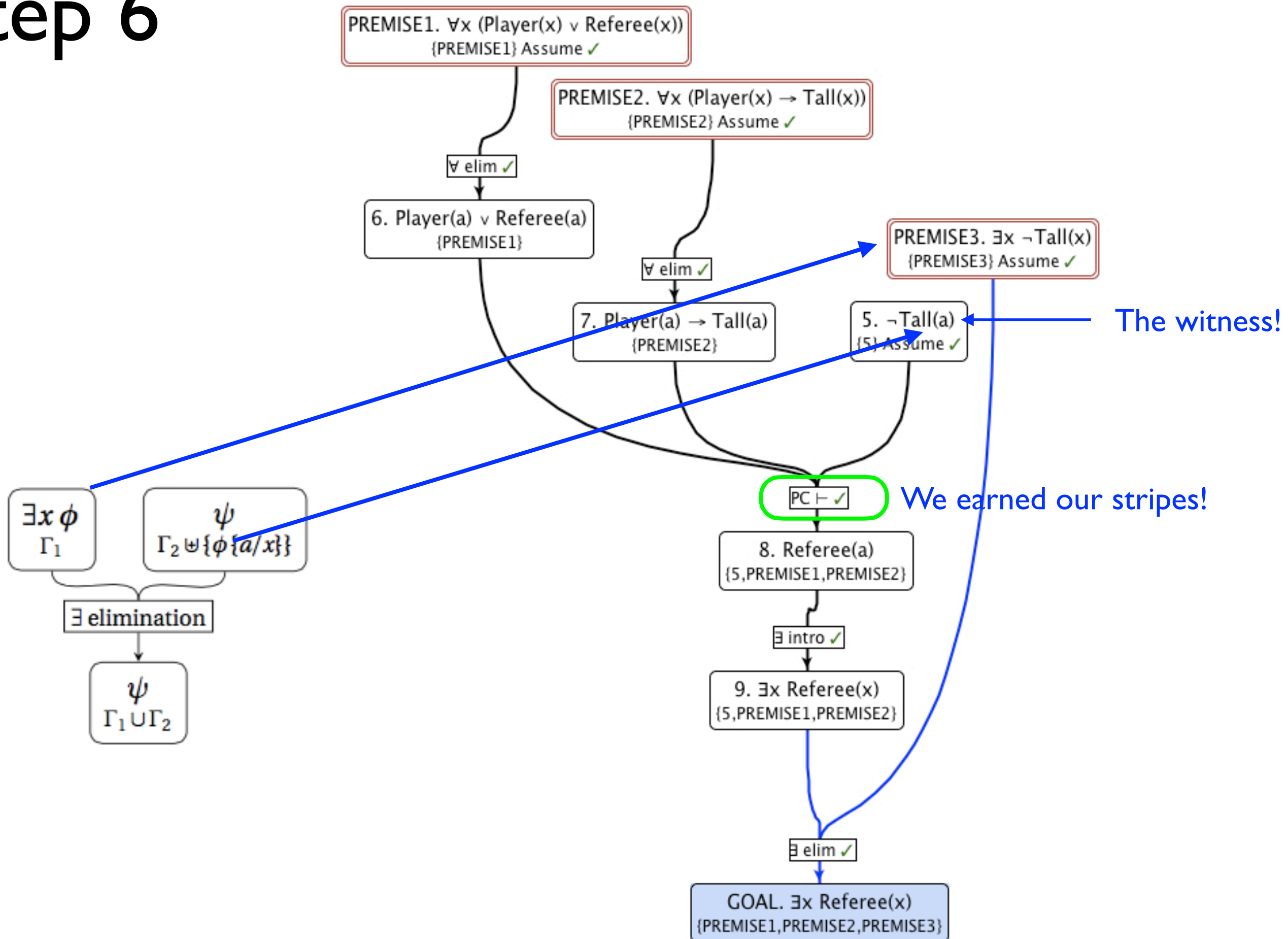
Step 6



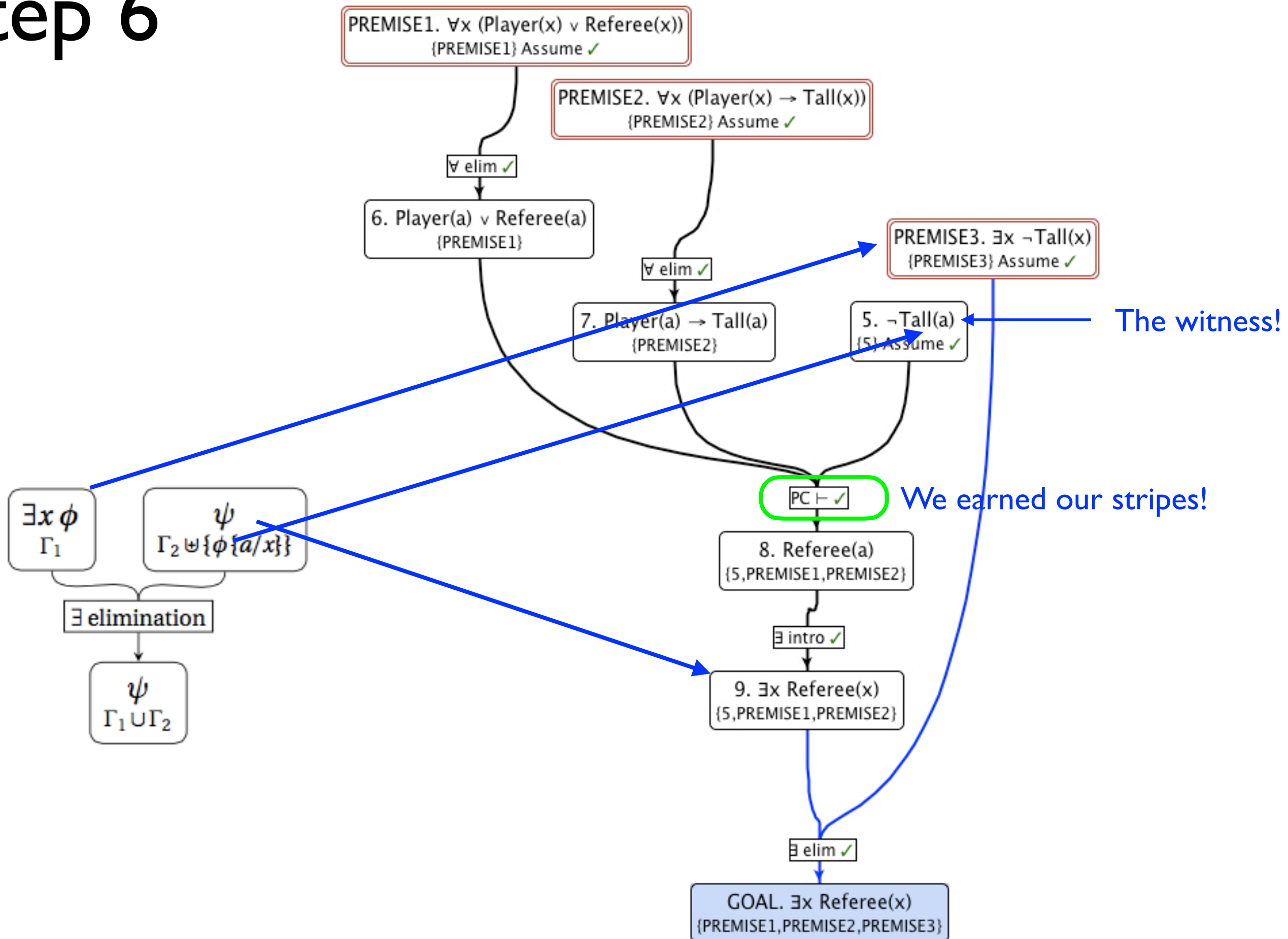
Step 6



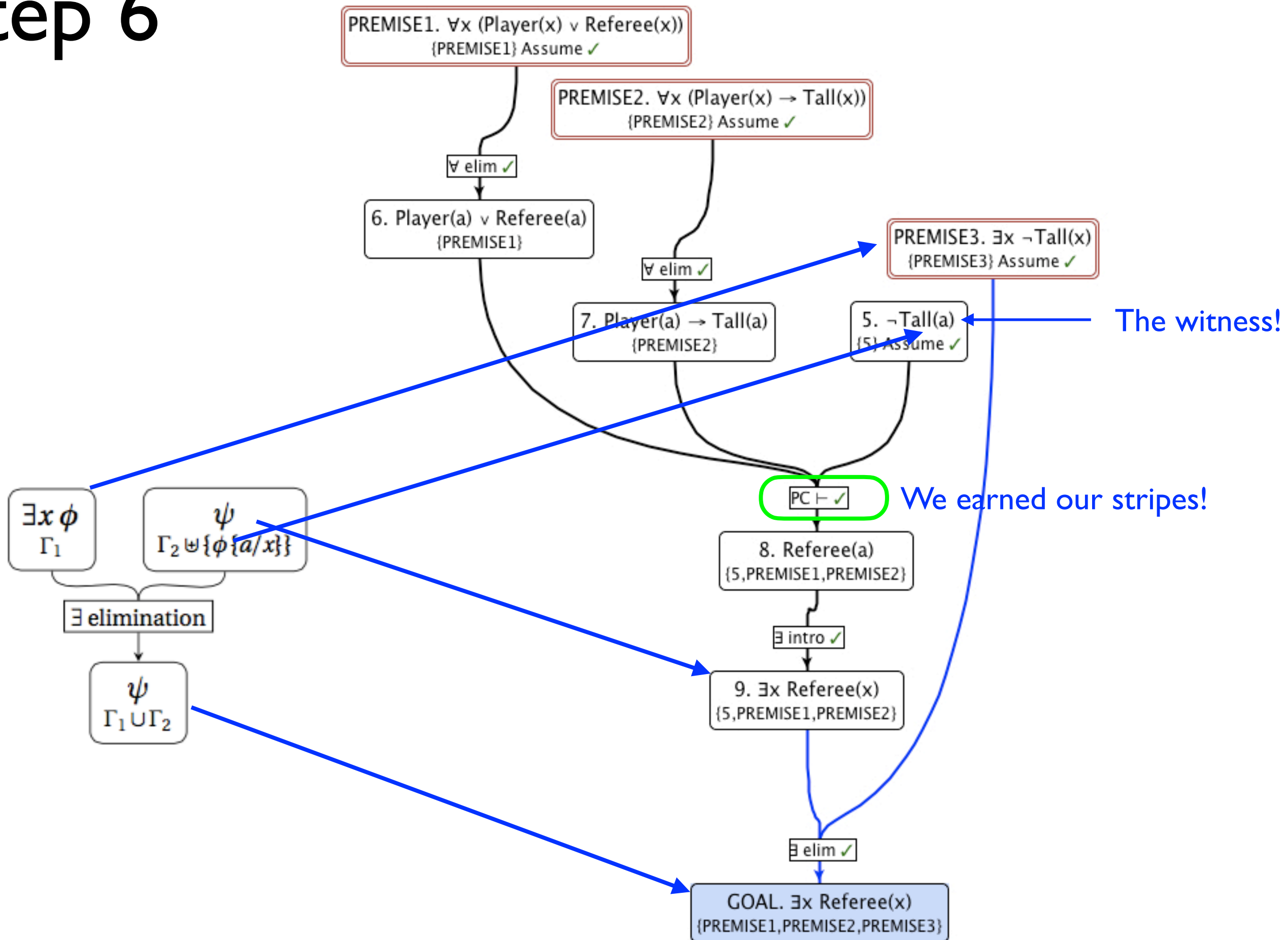
Step 6



Step 6



Step 6



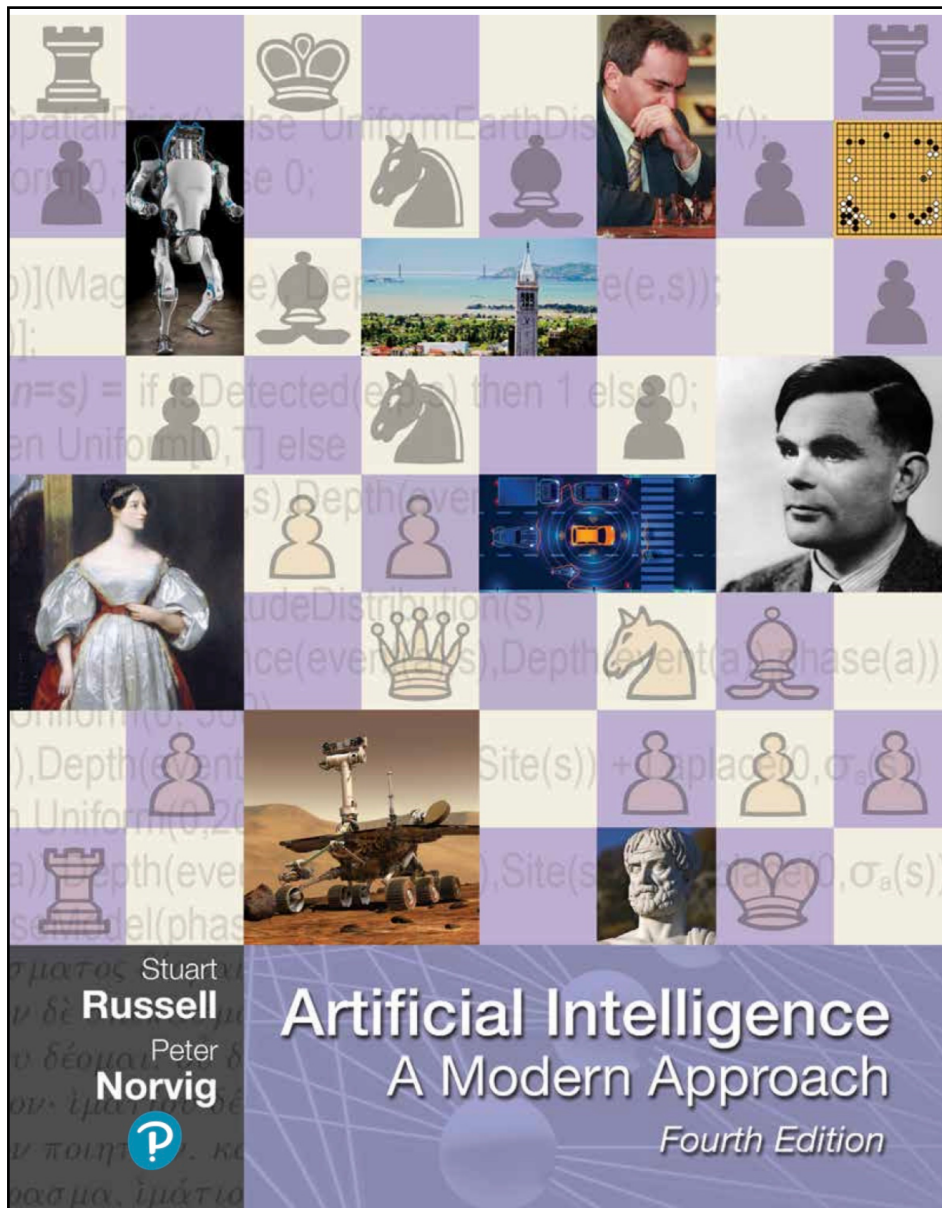
$$\{\forall x(\text{Scared}(x) \leftrightarrow \text{Small}(x)), \exists x \neg \text{Scared}(x)\} \vdash \exists x \neg \text{Small}(x)$$

$$\{\exists x, y \text{Contiguous}(x, y), \forall x, y(\text{Contiguous}(x, y) \rightarrow \neg \text{SameCountry}(x, y))\} \vdash \exists x, y \neg \text{SameCountry}(x, y)$$

Colonel West is a Criminal

<https://rpi.logicamodernapproach.com/hyperslate/public/Selmer.Bringsjord@gmail.com/WestIsACriminal>

Colonel West is a Criminal



<https://rpi.logicamodernapproach.com/hyperslate/public/Selmer.Bringsjord@gmail.com/WestIsACriminal>

Colonel West is a Criminal

9

INFERENCE IN FIRST-ORDER LOGIC

In which we define effective procedures for answering questions posed in first-order logic.

Chapter 7 defined the notion of **inference** and showed how sound and complete inference can be achieved for propositional logic. In this chapter, we extend those results to obtain algorithms that can answer any answerable question stated in first-order logic. This is significant, because more or less anything can be stated in first-order logic if you work hard enough at it.

Section 9.1 introduces inference rules for quantifiers and shows how to reduce first-order inference to propositional inference, albeit at great expense. Section 9.2 describes the idea of **unification**, showing how it can be used to construct inference rules that work directly with first-order sentences. We then discuss three major families of first-order inference algorithms: **forward chaining** and its applications to **deductive databases** and **production systems** are covered in Section 9.3; **backward chaining** and **logic programming** systems are developed in Section 9.4; and resolution-based **theorem-proving** systems are described in Section 9.5. In general, one tries to use the most efficient method that can accommodate the facts and axioms that need to be expressed. Reasoning with fully general first-order sentences using resolution is usually less efficient than reasoning with definite clauses using forward or backward chaining.

9.1 PROPOSITIONAL VS. FIRST-ORDER INFERENCE

This section and the next introduce the ideas underlying modern logical inference systems. We begin with some simple inference rules that can be applied to sentences with quantifiers to obtain sentences without quantifiers. These rules lead naturally to the idea that *first-order* inference can be done by converting the knowledge base to *propositional* logic and using *propositional* inference, which we already know how to do. The next section points out an obvious shortcut, leading to inference methods that manipulate first-order sentences directly.

Colonel West is a Criminal

9 INFERENCE IN FIRST-ORDER LOGIC

In which we define effective procedures for answering questions posed in first-order logic.

Chapter 7 defined the notion of **inference** and showed how sound and complete inference can be achieved for propositional logic. In this chapter, we extend those results to obtain algorithms that can answer any answerable question stated in first-order logic. This is significant, because more or less anything can be stated in first-order logic if you work hard enough at it.

Section 9.1 introduces inference rules for quantifiers and shows how to reduce first-order inference to propositional inference, albeit at great expense. Section 9.2 describes the idea of **unification**, showing how it can be used to construct inference rules that work directly with first-order sentences. We then discuss three major families of first-order inference algorithms: **forward chaining** and its applications to **deductive databases** and **production systems** are covered in Section 9.3; **backward chaining** and **logic programming** systems are developed in Section 9.4; and resolution-based **theorem-proving** systems are described in Section 9.5. In general, one tries to use the most efficient method that can accommodate the facts and axioms that need to be expressed. Reasoning with fully general first-order sentences using resolution is usually less efficient than reasoning with definite clauses using forward or backward chaining.

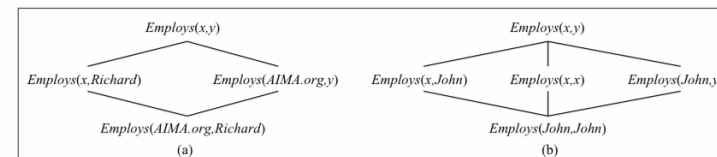
9.1 PROPOSITIONAL VS. FIRST-ORDER INFERENCE

This section and the next introduce the ideas underlying modern logical inference systems. We begin with some simple inference rules that can be applied to sentences with quantifiers to obtain sentences without quantifiers. These rules lead naturally to the idea that *first-order inference* can be done by converting the knowledge base to *propositional* logic and using *propositional* inference, which we already know how to do. The next section points out an obvious shortcut, leading to inference methods that manipulate first-order sentences directly.

272

280

Chapter 9. Inference in First-Order Logic



9.3 FORWARD CHAINING

A forward-chaining algorithm for propositional definite clauses was given in Section 7.5. The idea is simple: start with the atomic sentences in the knowledge base and apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made. Here, we explain how the algorithm is applied to first-order definite clauses and how it can be implemented efficiently. Definite clauses such as $\text{Situation} \Rightarrow \text{Response}$ are especially useful for systems that make inferences in response to newly arrived information. Many systems can be defined this way, and reasoning with forward chaining can be much more efficient than resolution theorem proving. Therefore it is often worthwhile to try to build a knowledge base using only definite clauses so that the cost of resolution can be avoided.

First-order definite clauses

First-order definite clauses closely resemble propositional definite clauses (page 217): they are disjunctions of literals of which *exactly one is positive*. A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. The following are first-order definite clauses:

$\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x).$
 $\text{King}(\text{John}).$
 $\text{Greedy}(y).$

Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified. (Typically, we omit universal quantifiers when writing definite clauses.) Definite clauses are a suitable normal form for use with Generalized Modus Ponens.

Not every knowledge base can be converted into a set of definite clauses, because of the single-positive-literal restriction, but many can. Consider the following problem:

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

especially useful for systems that make inferences in response to newly arrived information. Many systems can be defined this way, and reasoning with forward chaining can be much more efficient than resolution theorem proving. Therefore it is often worthwhile to try to build a knowledge base using only definite clauses so that the cost of resolution can be avoided.

First-order definite clauses

First-order definite clauses closely resemble propositional definite clauses (page 217): they are disjunctions of literals of which *exactly one is positive*. A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. The following are first-order definite clauses:

$$\begin{aligned} &King(x) \wedge Greedy(x) \Rightarrow Evil(x) . \\ &King(John) . \\ &Greedy(y) . \end{aligned}$$

Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified. (Typically, we omit universal quantifiers when writing definite clauses.) Definite clauses are a suitable normal form for use with Generalized Modus Ponens.

Not every knowledge base can be converted into a set of definite clauses, because of the single-positive-literal restriction, but many can. Consider the following problem:

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

especially useful for systems that make inferences in response to newly arrived information. Many systems can be defined this way, and reasoning with forward chaining can be much more efficient than resolution theorem proving. Therefore it is often worthwhile to try to build a knowledge base using only definite clauses so that the cost of resolution can be avoided.

First-order definite clauses

First-order definite clauses closely resemble propositional definite clauses (page 217): they are disjunctions of literals of which *exactly one* is positive, or is an implication whose antecedent is a conjunction of literals and whose consequent is a single positive literal. The following are

$$\begin{aligned} & King(x) \wedge Greedy(x) \Rightarrow Evil(x) . \\ & King(John) . \\ & Greedy(y) . \end{aligned}$$

Unlike propositional literals, first-order literals have variables. (The variables are assumed to be universally quantified when writing definite clauses.) Definite clauses are closed under the rule of Generalized Modus Ponens.

Not every knowledge base can be converted to a single-positive-literal restriction, but many can.

The law says that it is a crime for an American country Nono, an enemy of America, has some missiles, and it was sold to it by Colonel West, who is American.

We will prove that West is a criminal. First, we will represent these facts as first-order definite clauses. The next section shows how the forward-chaining algorithm solves the problem.

“... it is a crime for an American to sell weapons to hostile nations”:

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x) . \quad (9.3)$$

“Nono ... has some missiles.” The sentence $\exists x. Owns(Nono, x) \wedge Missile(x)$ is transformed into two definite clauses by Existential Elimination, introducing a new constant M_1 :

$$Owns(Nono, M_1) \quad (9.4)$$

$$Missile(M_1) \quad (9.5)$$

“All of its missiles were sold to it by Colonel West”:

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono) . \quad (9.6)$$

We will also need to know that missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x) \quad (9.7)$$

and we must know that an enemy of America counts as “hostile”:

$$Enemy(x, America) \Rightarrow Hostile(x) . \quad (9.8)$$

“West, who is American ...”:

$$American(West) . \quad (9.9)$$

“The country Nono, an enemy of America ...”:

$$Enemy(Nono, America) . \quad (9.10)$$

DATALOG

This knowledge base contains no function symbols and is therefore an instance of the class of **Datalog** knowledge bases—that is, sets of first-order definite clauses with no function symbols. We will see that the absence of function symbols makes inference much easier.

A simple forward-chaining algorithm

RENAMING

The first forward chaining algorithm we will consider is a very simple one, as shown in Figure 9.3. Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added. Notice that a fact is not “new” if it is just a **renaming** of a known fact. One sentence is a renaming of another if they are identical except for the names of the variables. For example, $Likes(x, IceCream)$ and $Likes(y, IceCream)$ are renamings of each other because they differ only in the choice of x or y ; their meanings are identical: everyone likes ice cream.

We will use our crime problem to illustrate how FOL-FC-ASK works. The implication sentences are (9.3), (9.6), (9.7), and (9.8). Two iterations are required:

- On the first iteration, rule (9.3) has unsatisfied premises. Rule (9.6) is satisfied with $\{x/M_1\}$, and $Sells(West, M_1, Nono)$ is added. Rule (9.7) is satisfied with $\{x/M_1\}$, and $Weapon(M_1)$ is added. Rule (9.8) is satisfied with $\{x/Nono\}$, and $Hostile(Nono)$ is added.

We will prove that West is a criminal. First, we will represent these facts as first-order definite clauses. The next section shows how the forward-chaining algorithm solves the problem.

“... it is a crime for an American to sell weapons to hostile nations”:

$$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x) . \quad (9.3)$$

“Nono ... has some missiles.” The sentence $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$ is transformed into two definite clauses by Existential Elimination, introducing a new constant M_1 :

$$\text{Owns}(\text{Nono}, M_1) \quad (9.4)$$

$$\text{Missile}(M_1) \quad (9.5)$$

“All of its missiles were sold to it by Colonel West”:

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) . \quad (9.6)$$

We will also need to know that missiles are weapons:

$$\text{Missile}(x) \Rightarrow \text{Weapon}(x) \quad (9.7)$$

and we must know that an enemy of America counts as “hostile”:

$$\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) . \quad (9.8)$$

“West, who is American ...”:

$$\text{American}(\text{West}) . \quad (9.9)$$

“The country Nono, an enemy of America ...”:

$$\text{Enemy}(\text{Nono}, \text{America}) . \quad (9.10)$$

This knowledge base contains no function symbols and is therefore an instance of the class of **Datalog** knowledge bases—that is, sets of first-order definite clauses with no function symbols. We will see that the absence of function symbols makes inference much easier.

A simple forward-chaining algorithm

The first forward chaining algorithm we will consider is a very simple one, as shown in Figure 9.3. Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added. Notice that a fact is not “new” if it is just a **renaming** of a known fact. One sentence is a renaming of another if

Colonel West is a Criminal

<https://rpi.logicamodernapproach.com/hyperslate/public/Selmer.Bringsjord@gmail.com/WestIsACriminal>

Let's explore it, approach understanding of it, and OpenAI's work, via an example ...

arXiv:2610.08144v1 [math.AP] 6 Oct 2026

NAVIER-STOKES LOST IN TRANSLATION

WHY LEAN VERIFICATION OF AI AUTOFORMALISATION DOES NOT GUARANTEE CORRECT NATURAL LANGUAGE PROOFS

A. BASTOUNIS, F. CIRCELLI, AND A. C. HANSEN*

ABSTRACT. Autoformalisation is increasingly used to verify mathematical texts, including those generated by AI, as in OpenAI's announced proof of blow-up of solutions to the Navier-Stokes equations. In this process, an AI system translates the text from a natural language (NL) into a formal language such as Lean. Once this translation is done, the argument expressed in the formal language can easily be mechanically verified. The purpose of this article is to demonstrate why this process may offer no confidence in the original NL argument, owing to the various difficulties in performing the translation semantically faithfully. In particular, we highlight that the problem of resolving ambiguities in mathematical NL text, which is necessary in order to provide semantically faithful translation, is arbitrarily high up in the Solvability Complexity Index (SCI) hierarchy/arithmetic hierarchy (the $SCI = \infty$). Hence, informally, providing semantically faithful AI autoformalisation is harder than any computational problem including the Halting problem (which has $SCI = 1$). To demonstrate the effect of this result we provide several examples of AI mistranslations of NL statements and proofs into Lean in practice, resulting in mismatches between NL proofs and their Lean 'verifications'. These include OpenAI's announced Navier-Stokes proof. In particular, we show that the formalised Lean proof does not correspond to the NL proof of blow-up of solutions to the Navier-Stokes equations.

1. INTRODUCTION: TWO TYPES OF AUTOFORMALISATION

We start by describing the two main types of autoformalisation from a natural language (NL) to a formal language:

- (i) **Translating an alleged proof in an NL to obtain a formal Lean proof.** Suppose a theorem (the statement, not including a proof) has already been faithfully and correctly translated into Lean. Suppose also the theorem is accompanied by an alleged NL proof. The goal is to let a machine (which we will call AI) translate the NL proof into Lean to obtain a formal proof of the theorem. Success is declared when the AI has produced a 'translation' which Lean accepts as a valid proof (meaning it compiles), with no use of Lean's 'sorry' keyword or extra axioms being permitted.
- (ii) **Faithful semantic translation of a mathematical text.** Suppose we are given a mathematical text (e.g. a book or a paper) written in an NL. The goal is to let an AI translate every definition, theorem, proposition, proof etc. in the text to Lean code that compiles, see e.g. the example in §5.1. The definitions, statements and proofs must themselves be translated, semantically preserving the source's mathematical content. For a book, this obligation applies throughout and includes the dependencies between concepts and results. If the stated goal is to validate its proofs, correspondence to the written arguments must also be established. In particular, a different, provable assertion would not constitute a semantically faithful translation. This autoformalisation problem is much harder than problem (i) above (see §4).

Warning: Meeting the success criterion of (i) does not mean that the original NL proof is correct. Indeed, a full Lean compilation of the 'translation' does not guarantee a semantically faithful translation. The AI could very easily change the semantics of the translation in order to satisfy this success criterion. For example, the

2020 *Mathematics Subject Classification.* 35Q30, 03Dxx (primary) and 68V20, 68Txx, 03B65 (secondary).

Key words and phrases. AI Autoformalisation, Navier-Stokes equations, Solvability Complexity Index hierarchy, boundaries of AI, Hilbert's 10th problem.

*Corresponding author: a.hansen@damtp.cam.ac.uk.

<https://arxiv.org/abs/2610.08144>

Bocardo2Quant; Questions?

<https://world.logicamodernapproach.com/hyperslate/public/Selmer.Bringsjord@gmail.com/Bocardo2Quant>

Bocardo; Bocardo $|Q| = 3$

Some As are not Bs.

$$\exists x [A(x) \wedge \neg B(x)]$$

All As are Cs.

$$\forall x [A(x) \rightarrow C(x)]$$

$\therefore$ Some Cs are not Bs.

$$\therefore \exists x [C(x) \wedge \neg B(x)]$$

Some As that v some Ds are not Bs.

$$\exists x \exists y [A(x) \wedge V(x, y) \wedge D(y) \wedge \neg B(x)]$$

All As that v some Ds are Cs.

$$\forall x [(A(x) \wedge \exists y (V(x, y) \wedge D(y))) \rightarrow C(x)]$$

Some Cs are not Bs.

$$\exists x [C(x) \wedge \neg B(x)]$$

Some As that v some Ds are not Bs that v' all Es.

$$\exists x \exists y [A(x) \wedge V(x, y) \wedge D(y) \wedge \neg(B(x) \wedge \forall z (V'(x, z) \wedge E(x)))]$$

All As that v some Ds are Cs.

$$\forall x [(A(x) \wedge \exists y (V(x, y) \wedge D(y))) \rightarrow C(x)]$$

Some Cs are not Bs that v' all Es.

$$\exists x [C(x) \wedge \neg(B(x) \wedge \forall z (E(z) \rightarrow V'(x, z)))]$$

*Hvis du forstår det, kan
du bevise det.*