Provadores de Teoremas e suas Aplicações

Marcus Vinícius Midena Ramos

PUC/UNIVASF

16/04/2019

marcus.ramos@univasf.edu.br
(15 de abril de 2019, 23:15)

Roteiro

Apresentação

Marcus Vinícius Midena Ramos

- Engenheiro Eletricista (EPUSP 1982);
- Mestre em Sistemas Digitais (EPSUP 1991);
- Doutor em Ciência da Computação (UFPE 2016);
- Professor do curso de Engenharia de Computação da UNIVASF (desde 04/2018); antes EPUSP, Sumaré, FASP, Senac, PUC e Maurício de Nassau;
- Coautor do livro Linguagens Formais (com I.S. Vega e J.J. Neto, Bookman 2009);
- Professor das disciplinas: Teoria da Computação, Linguagens Formais e Autômatos e Compiladores;
- Coordenador do grupo de estudos Provadores de Teoremas e suas Aplicações (desde 07/2018)
- Trabalha com a formalização matemática de linguagens livres de contexto usando o Coq (desde 2013).

Motivação

Sobre o que vamos falar?

- Matemática formal;
- Prova interativa de teoremas;
- Desenvolvimento interativo de programas certificados;
- Assistentes de prova em geral;
- Coq em particular.

Objetivos

- Introduzir os Assistentes de Prova Interativos (também conhecidos como Provadores de Teoremas);
- Discutir o seu papel no desenvolvimento de programas e na prova de teoremas;
- Apresentar alguns projetos de formalização relevantes, tanto na indústria quanto na academia;
- Apresentar tópicos das principais teorias utilizadas;
- Apresentar o assistente de provas Coq;
- Mostrar um exemplo completo.

Provadores de Teoremas × Assistentes de Prova Interativos

Termos diferentes para designar a mesma coisa:

- "Provadores de Teoremas" é um termo bastante usado mas que não corresponde à realidade das ferramentas:
  Os provadores não provam teoremas, pelo menos não sozinhos;
- Por isso, este é considerado um termo um pouco mais pretensioso (ou ambicioso);
- Neste sentido, o termo "Assistentes Interativos de Provas" é mais razoável:
  Os assistentes ajudam o usuário a construir provas e não tem como objetivo construí-las sozinhos;
- Ainda assim, os Assistentes Interativos de Provas oferecem alguns recursos de automação que servem para casos especiais;
- Nesta apresentação, assim como na maior parte da literatura especializada, os dois termos serão usados de forma indistinta.

História e prática corrente

- Provas de teoremas:
  - Informais;
  - Difíceis de construir;
  - Difíceis de verificar.
- Programas de computador:
  - Informais;
  - Difíceis de construir;
  - Difíceis de testar.
- Coincidência?

História e prática corrente

- **NA VERDADE NÃO**, já que a prova de teoremas e o desenvolvimento de software possuem essencialmente a mesma natureza;
- De acordo com a Correspondência de Curry-Howard, desenvolver um programa é a mesma coisa que provar um teorema, e vice-versa;
- Explorar essa similaridade pode ser benéfica para ambas as atividades:
  - Raciocínio ("reasoning") pode ser introduzido na programação, e
  - Computação pode ser usada na prova de teoremas.
- Como tirar proveito de tudo isso então?

Perspectivas

- A formalização matemática (*"matemática codificada no computador"*) é a resposta;
- Raciocínio auxliado por computador;
- Uso de assistentes interativos de provas (provadores de teoremas).

Questões iniciais

- O que são teoremas?
- O que são provas?
- O que são provadores de teoremas?

Questões iniciais

Perguntar não ofende:

- Coisa de maluco?
- Tem aplicação prática?
- Por que eu deveria me interessar por isso?

Questões iniciais

Nova (!??) maneira de:

- Provar teoremas;
- Desenvolver software.

Questões iniciais

Provadores de Teoremas:

- Programa de computador;
- Existem vários disponíveis;
- Mudam a teoria subjacente (por exemplo, clássica ou construtiva), as linguagens e as interfaces;
- Geralmente são gratuitos;
- Disponíveis para várias plataformas (Windows, Linux, iOS);
- Fazem a verificação mecânica da correção de uma prova;
- Funcionam como assistente interativo para elaboração de provas;
- Eventualmente permitem a extração de programas;
- Usam diversas linguagens e teorias.

Questões iniciais

Vantagens para:

- Matemáticos;
- Cientistas da computação;
- Programadores;
- Engenheiros de software;
- Empresas de desenvolvimento de software e hardware;
- Usuários de programas, componentes e aplicativos.

Questões iniciais

Para os matemáticos:

- Formalização;
- Segurança;
- Publicação;
- Compartilhamento;
- Reutilização.

Questões iniciais

- Prova de teoremas e desenvolvimento de software, o que uma coisa tem a ver com a outra?
- Não são coisas completamente diferentes? Teoria e prática?
- Tem tudo a ver!

Questões iniciais

Prova?

- Argumentação incontestável sobre a validade de uma proposição.
- Argumentação?
- Incontestável?
- Proposição?

Dependendo da audiência e das linguagens utilizadas, uma prova pode ou não ser aceita como válida. O desafio é propor uma linguagem que seja simples o suficiente para convencer tdas as audiências, incluindo e principalmente as máquinas.

Questões iniciais

Teorema?

- Proposição não-trivial acerca de alguma definição ou conjunto de definições.
- Não-trivial?
- Definição?

Exemplo: $\forall n, n^2 + n$ é par.

Questões iniciais

Teoria?

- Conjunto limitado de definições (uma ou mais);
- Conjunto, geralmente extenso, de teoremas (ou lemas) que dizem respeito à(s) definição(ões);
- Teoremas e lemas são propriedades não-triviais das definições e que, por causa disso, necessitam ser demonstradas (provadas);
- As provas precisam ser formuladas em algum tipo de cálculo;
- A simplicidade do cálculo pode permitir que as provas sejam verificadas automaticamente.

Exemplos: (i) Cálculo Lambda e (ii) Linguagens Livres de Contexto.

Questões iniciais

Provar teoremas e desenvolver sofware?

- Correspondência de Curry-Howard;
- Dedução Natural;
- Cálculo Lambda.

Questões iniciais

Se alguns requisitos forem observados, a prova de um teorema se torna o programa que atende à uma certa especificação.

- Provas $\Leftrightarrow$ Programas;
- Proposições (ou Tipos) $\Leftrightarrow$ Especificações.

Questões iniciais

Conseqüência prática:

- Provar um teorema é a mesma coisa que construir um programa!

Questões iniciais

Atividades usuais:

- Obter uma prova para um teorema, ou seja,
  Prova $\Rightarrow$ Teorema (Proposição);
- Construir um programa que atende uma especificação, ou seja,
  Programa $\Rightarrow$ Especificação (Tipo).

Questões iniciais

Correspondência de Curry-Howard:

- Provas são Programas;
- Programas são Provas;
- Proposições são Especificações;
- Especificações são Proposições.

Questões iniciais

Desta forma, podemos:

- Construir uma especificação para um programa na forma de uma proposição;
- Construir uma prova para esta proposição;
- Obter o programa a partir da prova.

Questões iniciais

Conseqüências:

- Programas certificados;
- Não há necessidade de testes;
- Corretos por construção;
- Maior confiabilidade.

Importância decorre do uso generalizado e crescente de sistemas computacionais, especialmente em aplicações que oferecem risco à vida ou ao patrimônio.

Questões iniciais

Requisitos:

- Conhecer Provadores de Teoremas;
- Conhecer a teoria subjacente;
- Experiência;
- Força de vontade.

Assistentes de Prova

Characteristics

- Software tools that assist the user in theorem proving and program development;
- First initiative dates from 1967 (Automath, De Bruijn);
- Many provers are available today (Coq, Agda, Mizar, HOL, Isabelle, Matita, Nuprl...);
- Check The Seventeen Provers of the World
- Interactive;
- Graphical interface;
- Proof/program checking;
- Proof/program construction.

The Seventeen Provers of the World

The Seventeen Provers of the World

Compiled by Freek Wiedijk
(and with a Foreword by Dana Scott)

<freek@cs.ru.nl>
Radboud University Nijmegen

Abstract. We compare the styles of several proof assistants for mathematics. We present Pythagoras' proof of the irrationality of $\sqrt{2}$ both informal and formalized in (1) *HOL*, (2) *Mizar*, (3) *PVS*, (4) *Coq*, (5) *Otter/Ivy*, (6) *Isabelle/Isar*, (7) *Alfa/Agda*, (8) *ACL2*, (9) *PhoX*, (10) *IMPS*, (11) *Metamath*, (12) *Theorema*, (13) *Lego*, (14) *Nuprl*, (15) *Ωmega*, (16) *B method*, (17) *Minlog*.

Usage

1. The user writes a statement (proposition) or a type expression (specification) in the language of the underlying logic;
2. He constructs (directly or indirectly):
   - A proof of the theorem;
   - A program (term) that complies to the specification.
3. Directly: the proof/term is written in the formal language accepted by the assistant;
4. Indirectly: the proof/term is built with the assistance of an interactive "tactics" language:
5. In either case, the assistant checks that the proof/term complies to the theorem/specification.

Check and/or construct

- Proof assistants check that proofs/terms are correctly constructed;
- This is done via simple type-checking algorithms;
- Automated proof/term construction might exist is some cases, to some extent, but this is not the main focus;
- Thus the name “proof assistant”;
- Automated theorem proofing might be pursued, due to “proof irrelevance”;
- Automated program development, on the other hand, is unrealistic.

Main benefits

- Proofs and programs can be mechanically checked, saving time and effort and increasing reliability;
- Checking is efficient;
- Results can be easily stored and retrieved for use in different contexts;
- Tactics help the user to construct proofs/programs;
- User gets deeper insight into the nature of his proofs/programs, allowing further improvement.

Applications

- Formalization and verification of theorems and whole theories;
- Verification of computer programs;
- Correct software development;
- Automatic review of large and complex proofs submitted to journals;
- Verification of hardware and software components.

Drawbacks

- Failures in infrastructure may decrease confidence in the results (proof assistant code, language processors, operating system, hardware etc);
- Size of formal proofs;
- Reduced numer of people using proof assistants;
- Slowly increasing learning curve;
- Resemblance of computer code keeps pure mathematicians uninterested.

Aplicações

Questões iniciais

O mundo está mudando:

- Empresas de software estão usando Provadores de Teoremas;
- Elas estão contratando profissionais que sabem usá-los;
- Competitividade, produtividade e qualidade;
- Aplicações importantes;
- Mercado emergente.

Questões iniciais

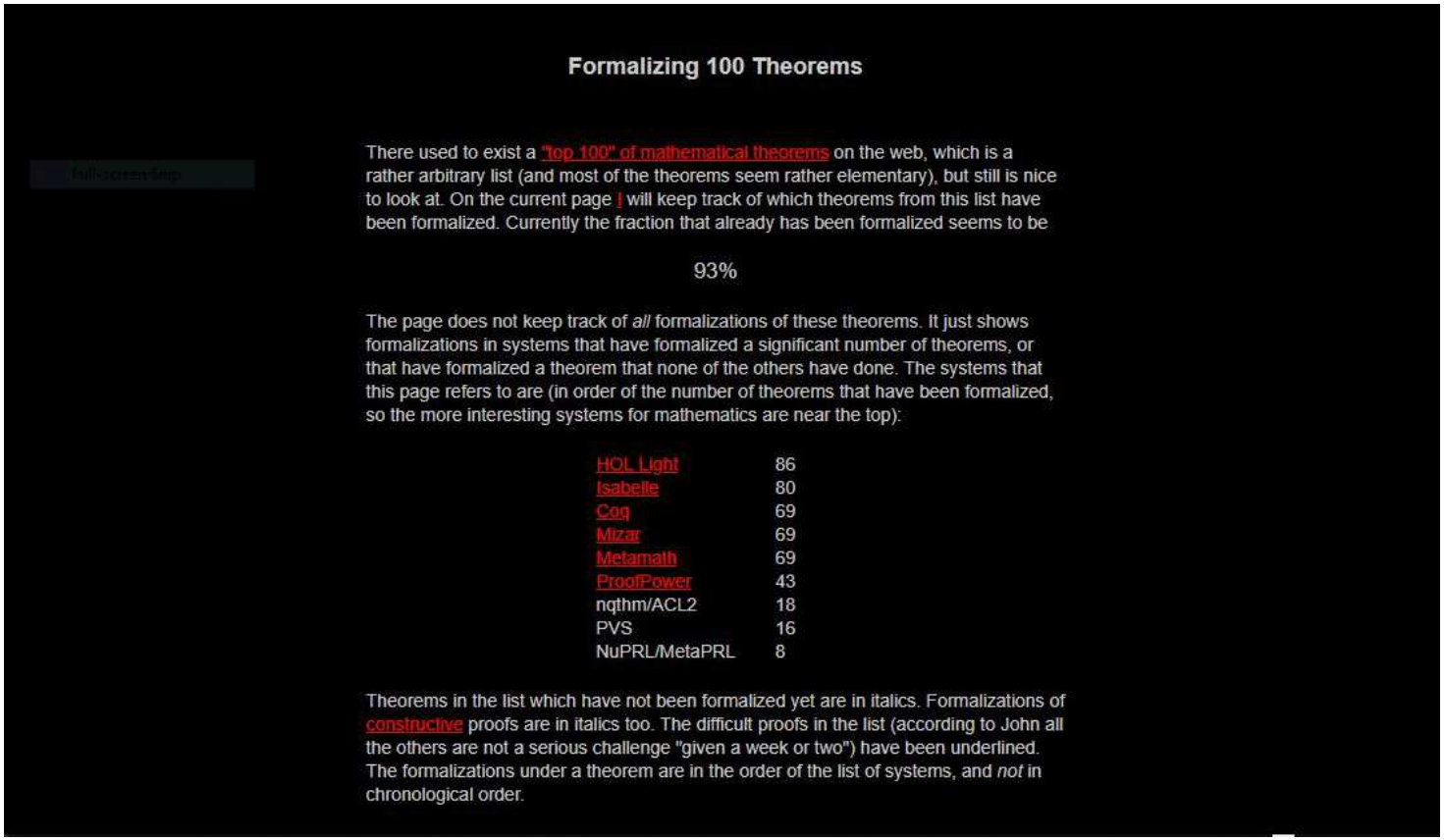
Já mudou alguma coisa?

- Intel;
- Microsoft;
- Compiladores, sistemas operacionais, chips, smart cards etc;
- Visível na Europa e nos EUA;
- Imperceptível no Brasil;
- Oportunidades de carreira e de empreendimento.

Introduction

- Great and increasing interest in formal proof and program development over the recent years;
- Main areas include:
  - Programming language semantics formalization;
  - Mathematics formalization;
  - Education.
- Important projects in both academy and industry;
- Top 100 theorems (93% formalized as of April/2019);
- Check 100 Theorems;
- One way road.

100 Theorems



Formalizing 100 Theorems

There used to exist a "top 100" of mathematical theorems on the web, which is a rather arbitrary list (and most of the theorems seem rather elementary), but still is nice to look at. On the current page I will keep track of which theorems from this list have been formalized. Currently the fraction that already has been formalized seems to be

93%

The page does not keep track of *all* formalizations of these theorems. It just shows formalizations in systems that have formalized a significant number of theorems, or that have formalized a theorem that none of the others have done. The systems that this page refers to are (in order of the number of theorems that have been formalized, so the more interesting systems for mathematics are near the top):

HOL Light	86
Isabelle	80
Coq	69
Mizar	69
Metamath	69
ProofPower	43
nqthm/ACL2	18
PVS	16
NuPRL/MetaPRL	8

Theorems in the list which have not been formalized yet are in italics. Formalizations of constructive proofs are in italics too. The difficult proofs in the list (according to John all the others are not a serious challenge "given a week or two") have been underlined. The formalizations under a theorem are in the order of the list of systems, and *not* in chronological order.

A Sampler of Formally Checked Projects

Some remarkable projects:

- Four Color Theorem;
- Odd Order Theorem;
- Kepler Conjecture;
- Homotopy Type Theory and Univalent Foundations of Mathematics;
- Compiler Certification;
- Microkernel Certification;
- Digital Security Certification.

Four Color Theorem

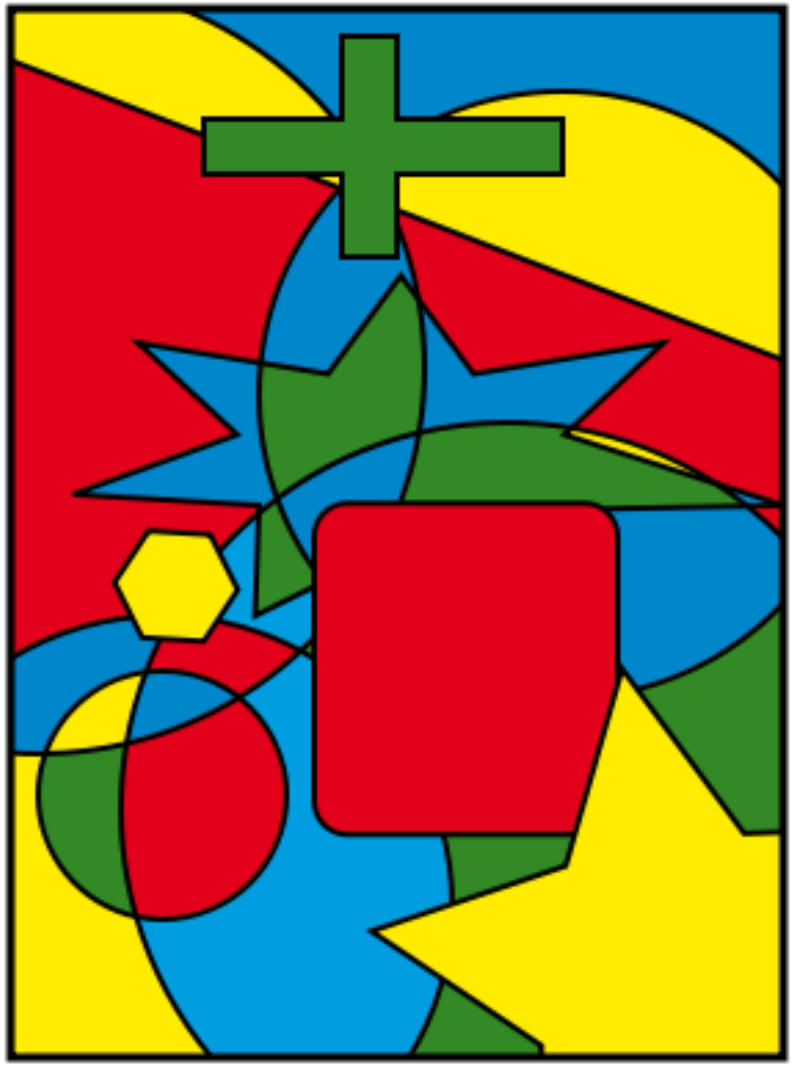
- Stated in 1852, proved in 1976 and again in 1995;
- The two proofs used computers to a some extent, but were not fully mechanized;
- In 2005, Georges Gonthier (Microsoft Research) and Benjamin Werner (INRIA) produced a proof script that was fully checked by a machine;
- Milestone in the history of computer assisted proofing;
- 60,000 lines of Coq script and 2,500 lemmas;
- Byproducts.

Four Color Theorem

According to Wikipedia:

> *"In mathematics, the four color theorem, or the four color map theorem, states that, given any separation of a plane into contiguous regions, producing a figure called a map, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color. Adjacent means that two regions share a common boundary curve segment, not merely a corner where three or more regions meet."*

Four Color Theorem



Four Color Theorem

"Although this work is purportedly about using computer programming to help doing mathematics, we expect that most of its fallout will be in the reverse direction using mathematics to help programming computers."

Georges Gonthier

Odd Order Theorem

- Also known as the Feit-Thomson Theorem;
- "In mathematics, the Feit–Thompson theorem, or odd order theorem, states that every finite group of odd order is solvable" (Wikipedia);
- Important to mathematics (in the classification of finite groups) and cryptography;
- Conjectured in 1911, proved in 1963;
- Formally proved by a team led by Georges Gonthier in 2012;
- Six years with full-time dedication;
- Huge achievement in the history of computer assisted proofing;
- 150,000 lines of Coq script and 13,000 theorems;

Opportunity

Fermat's Last Theorem

Statement in Coq:

```
Theorem Fermat: forall x y z n: nat, (x^n+y^n=z^n)->(n<=2).
```

According to Wikipedia:

- "In number theory Fermat's Last Theorem (sometimes called Fermat's conjecture, especially in older texts) states that no three positive integers a, b, and c satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than 2";
- "This theorem was first conjectured by Pierre de Fermat in 1637 in the margin of a copy of Arithmetica where he claimed he had a proof that was too large to fit in the margin. The first successful proof was released in 1994 by Andrew Wiles, and formally published in 1995, after 358 years of effort by mathematicians. The proof was described as a 'stunning advance' in the citation for his Abel Prize award in 2016".

Compiler Certification

- CompCert, a fully verified compiler for a large subset of C that generates PowerPC code;
- Object code is certified to comply with the source code in all cases;
- Applications in avionics and critical software systems;
- Not only checked, but also developed in Coq;
- Three persons-years over a five yers period;
- 42,000 lines of Coq code.

Microkernel Certification

- Critical component of operating systems, runs in privileged mode;
- Harder to test in all situations;
- seL4, written in C (10,000 lines), was fully checked in HOL/Isabelle;
- No crash, no execution of any unsafe operation in any situation;
- Proof is 200,000 lines long;
- 11 persons-years, can go down to 8, 100% overhead over a non-certified project.

Digital Security Certification

- JavaCard smart card platform;
- Personal data such as banking, credit card, health etc;
- Multiple applications by different companies;
- Confidence and integrity must be assured;
- Formalization of the behaviour and the properties of its components;
- Complete certification, highest level achieved;
- INRIA, Schlumberger and Gemalto.

Questões iniciais

O profissional do futuro precisa conhecer e saber usar a teoria. Provadores de Teoremas são apenas uma ferramenta.

Coq

Visão Geral

Coq é simultaneamente:

- Uma linguagem de programação funcional (que pode ser usada para construir programas e realizar computações);
- Um assistente interativo de provas que possibilita a extração de código;
- É justamente o uso combinado destes recursos que o torna uma ferramenta muito poderosa.

Visão Geral

Coq pode ser executado em linha de comando ou através de uma interface gráfica (CoqIDE ou Proof General).

- Coq implementa duas linguagens, Gallina e Vernacular;
- Gallina é a linguagem usada para representar termos (provas e programas) e proposições (teoremas e tipos);
- Gallina é baseada no Cálculo de Contruções com Definições Indutivas;
- Vernacular é a linguagem de comando para interação com o usuário.

Overview

- Developed by Huet/Coquand at INRIA in 1984;
- First version released in 1989, inductive types were added in 1991;
- Continuous development and increasing usage since then;
- The underlying logic is the Calculus of Constructions with Inductive Definitions;
- It is implemented by a typed functional programming with a higher order logic language called *Gallina*;
- Interaction with the user is via a command language called *Vernacular*;
- Constructive logic with large standard library and user contributions base;
- Extensible environment;
- Check Coq Home Page for downloads, documentation, communities and much more.

User session

The proof can be constructed directly ou indirectly. In the indirect case:

- The initial goal is the theorem or specification supplied by the user;
- The initial context of the proof is usually empty;
- The application of a "tactic", on either the current goal or one of the premises, substitutes the current goal for zero or more subgoals, or changes the context accordingly;
- This creates the notion of a stack of subgoals, all of which have to be proved in reverse order;
- The context changes and may incorporate new premises;
- The process is repeated for each subgoal, until no subgoal remains;
- The proof/expression is then constructed, checked and saved by the proof assistant from the sequence of tactics used.

Tactics usage

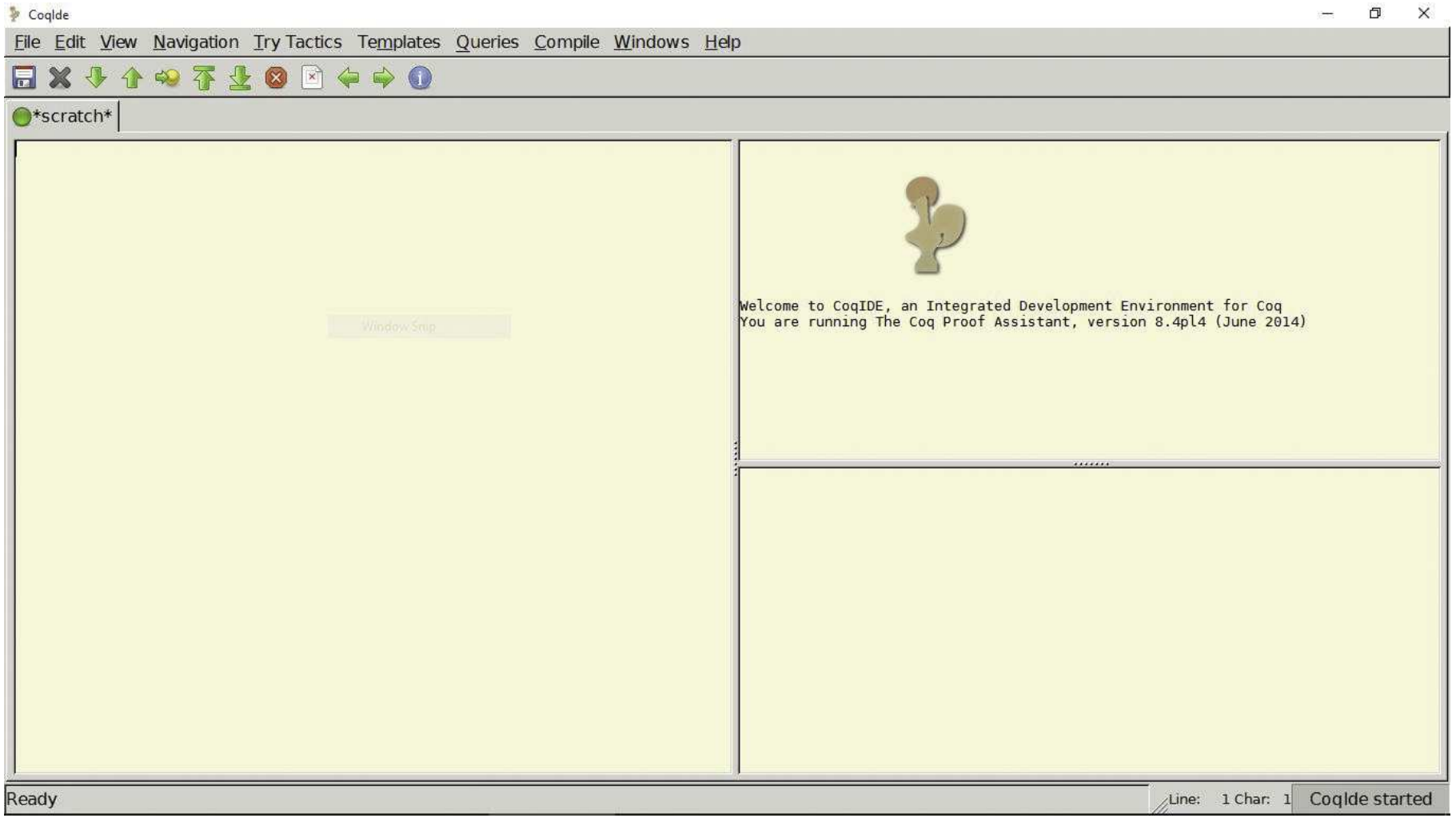
- Inference rules map premises to conclusions;
- *Forward reasoning* is the process of moving from premises to conclusions;
  - Example: from a proof of a and a proof of b one can prove $a \wedge b$;
- *Backward reasoning* is the process of moving from conclusions to premises;
  - Example: to prove $a \wedge b$ one has to prove a and also prove b;
- Coq uses *backward reasoning*;
- They are implemented by "tactics";
- A tactic reduces a goal to its subgoals, if any, changes the context or simply proves the goal.

CoqIDE

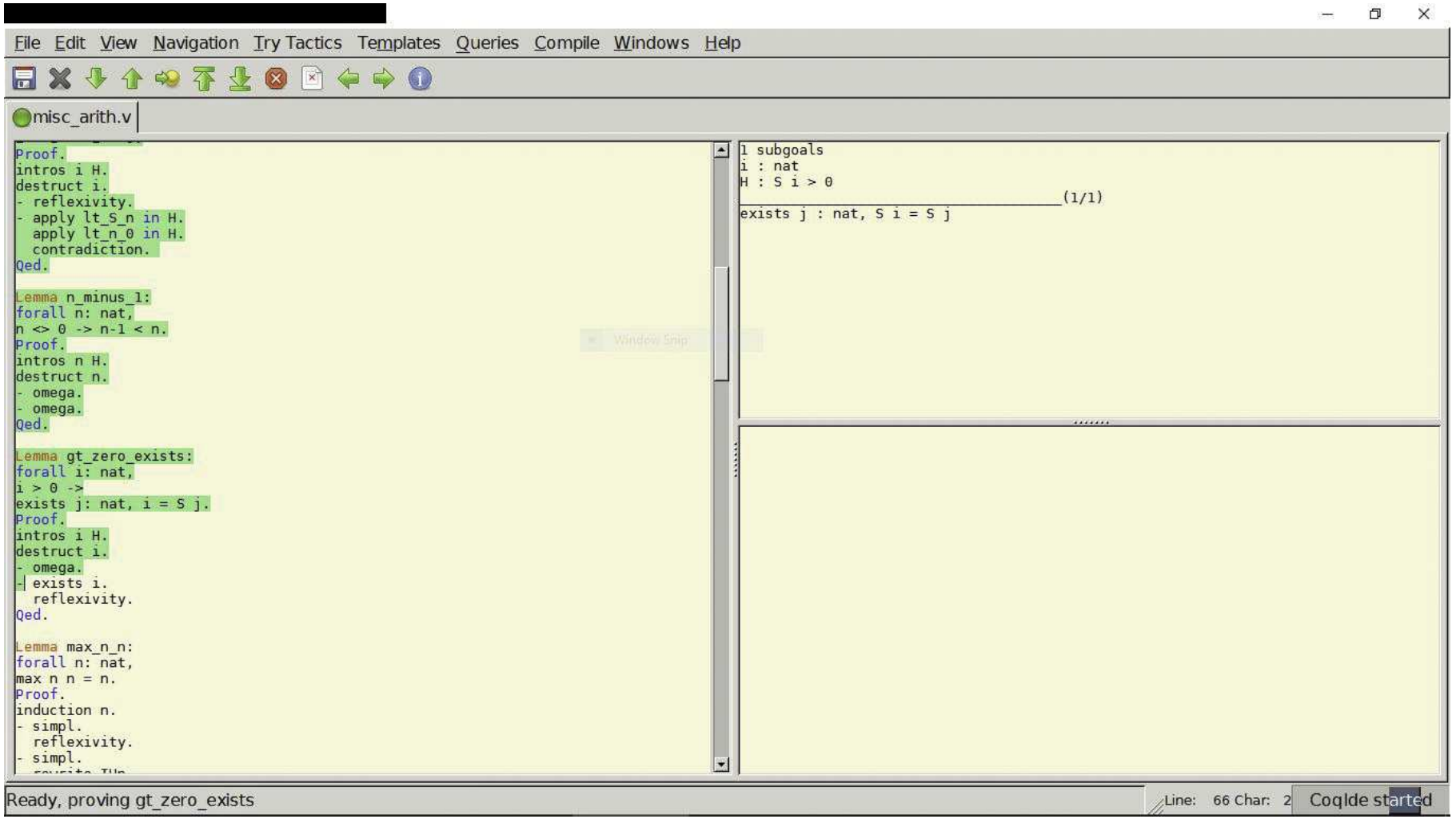
Uma interface gráfica para uso do Coq:

- Menus, atalhos e preferências;
- Lado esquerdo: editor de scripts (seqüência de definições e lemas/teoremas da teoria que se deseja formalizar);
- Lado direito em cima: contexto usado na prova ("goal" corrente e conjunto de premissas disponíveis);
- Lado direito embaixo: mensagens do sistema para o usuário.

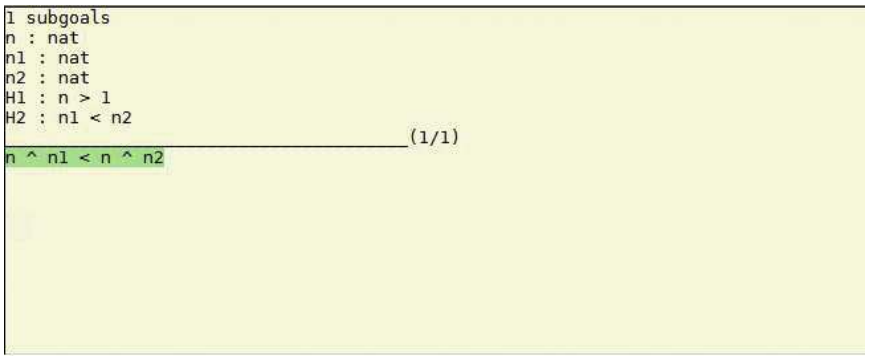
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Welcome to CoqIDE, an Integrated Development Environment for Coq
You are running The Coq Proof Assistant, version 8.4pl4 (June 2014)
Ready
Line: 1 Char: 1
CoqIde started

CoqIDE


File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
misc_arith.v
Proof.
intros i H.
destruct i.
- reflexivity.
- apply lt_S_n in H.
apply lt_n_0 in H.
contradiction.
Qed.
Lemma n_minus_1:
forall n: nat,
n <> 0 -> n-1 < n.
Proof.
intros n H.
destruct n.
- omega.
- omega.
Qed.
Lemma gt_zero_exists:
forall i: nat,
i > 0 ->
exists j: nat, i = S j.
Proof.
intros i H.
destruct i.
- omega.
- exists i.
reflexivity.
Qed.
Lemma max_n_n:
forall n: nat,
max n n = n.
Proof.
induction n.
- simpl.
reflexivity.
- simpl.
1 subgoals
i : nat
H : S i > 0
(1/1)
exists j : nat, S i = S j
Ready, proving gt_zero_exists
Line: 66 Char: 2
CoqIde started

CoqIDE



CoqIDE

Exemplo de sessão, prova da proposição:

```
Lemma example:
∀ a b c: Prop,
(a→ (b→ c))→ (b→ (a→ (b∧ c))).
```

- Seqüência de táticas;
- Pode-se avançar (ctrl-arrow-down) ou retroceder (ctrl-arrow-up) tática por tática;
- Táticas já processadas tornam-se verdes e ficam "travadas";
- Observe a mudança do "goal" e a geração de novos "subgoals";
- Observe a mudança do contexto.

CoqIDE

Script que constrói a prova:

```
Lemma example:
∀ a b c: Prop,
(a→ (b→ c))→ (b→ (a→ (b∧ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
  + exact H3.
  + exact H2.
Qed.
```

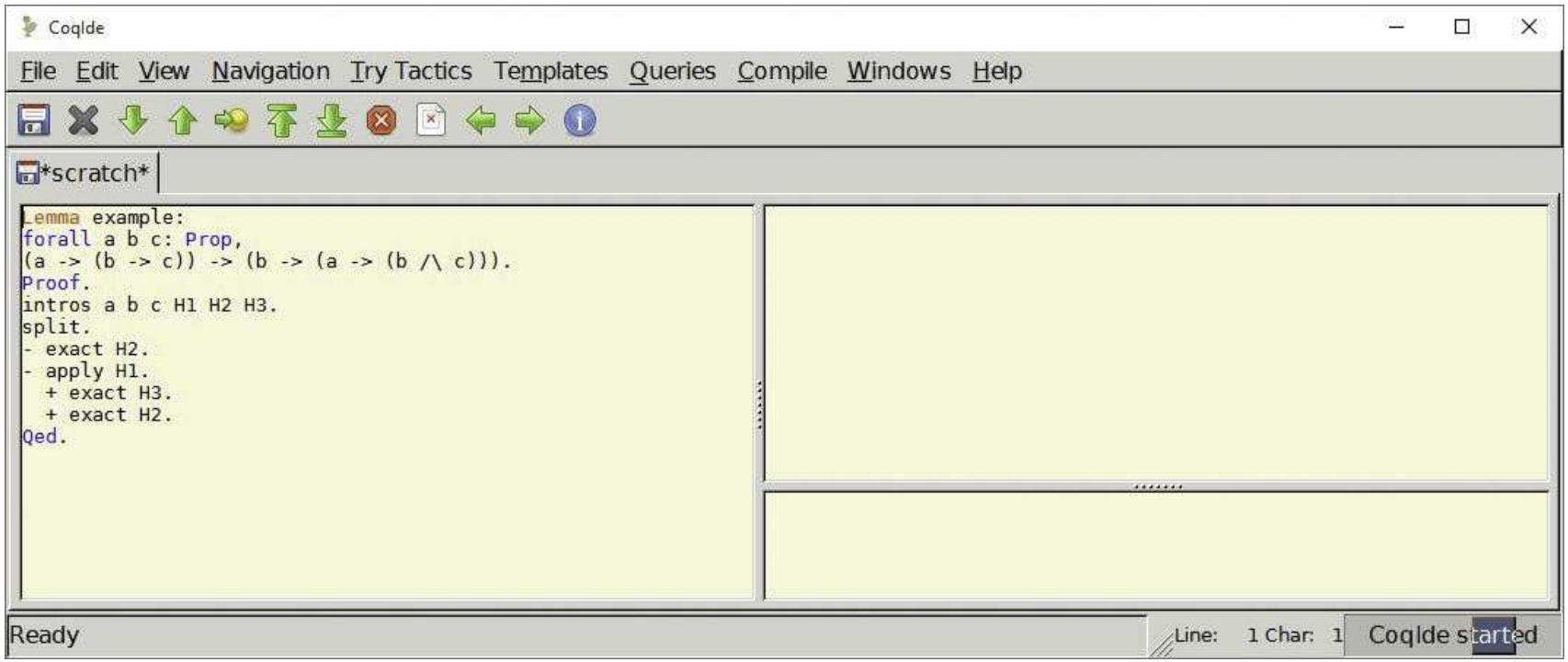
CoqIDE

Prova:

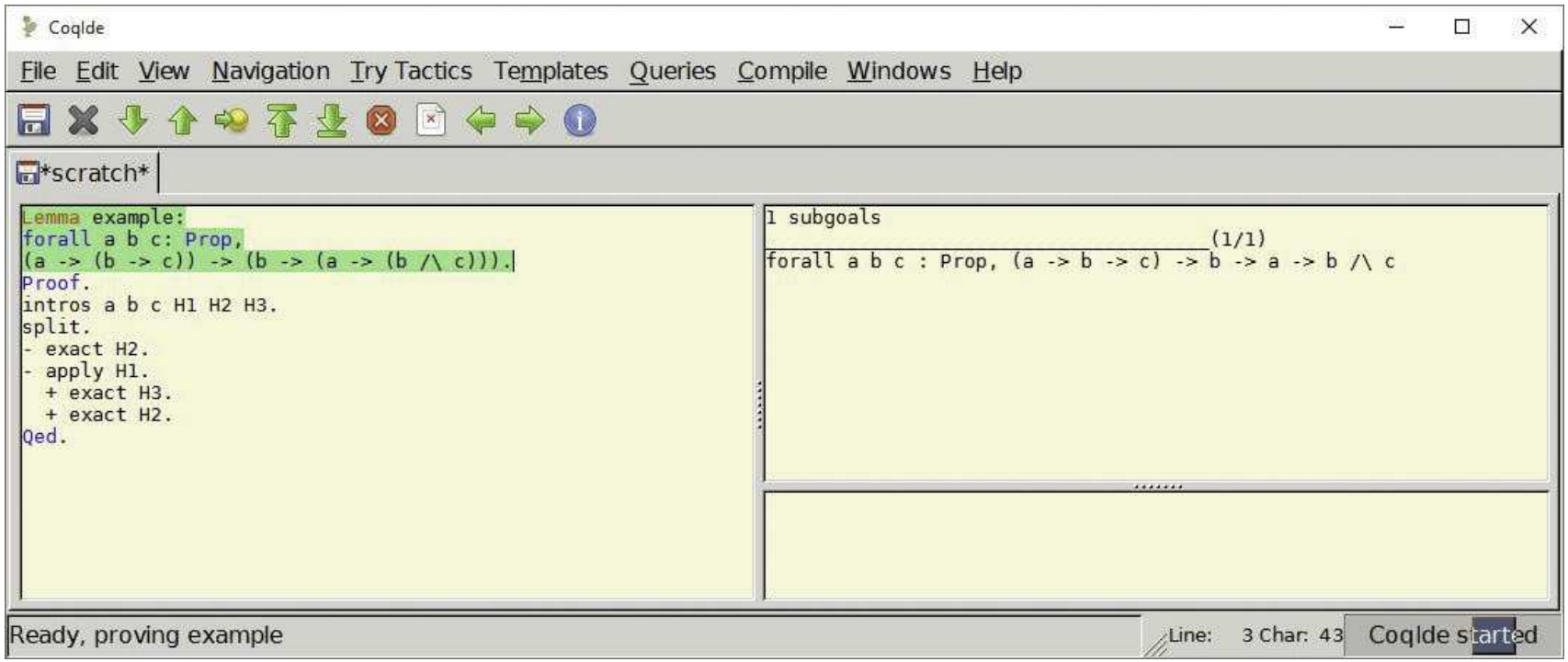
```
Print example.

example =
fun (a b c : Prop) (H1 : a → b → c) (H2 : b) (H3 : a)
⇒ conj H2 (H1 H3 H2)
: ∀ a b c : Prop, (a → b → c) → b → a → b ∧ c
```

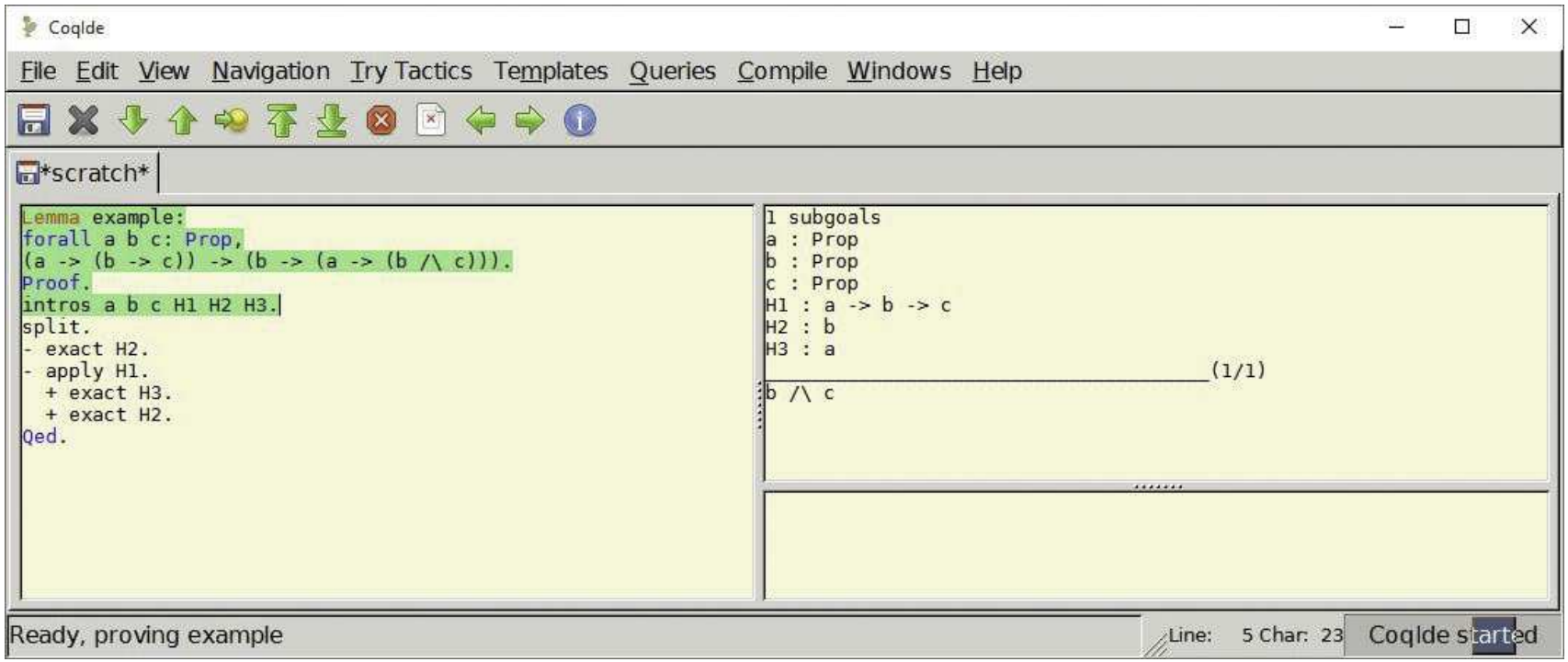
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
  + exact H3.
  + exact H2.
Qed.
Ready
Line: 1 Char: 1
CoqIde started

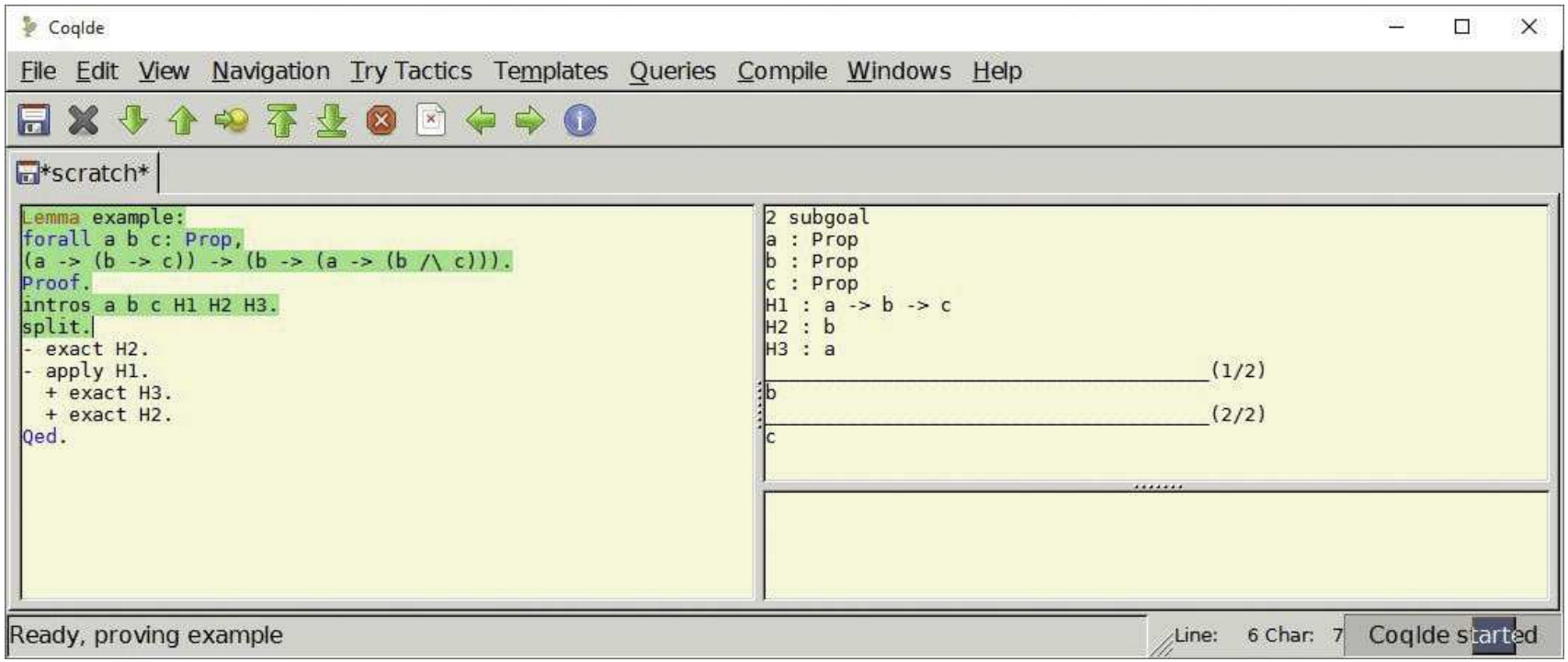
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
1 subgoals
(1/1)
forall a b c : Prop, (a -> b -> c) -> b -> a -> b /\ c
Ready, proving example
Line: 3 Char: 43
CoqIde started

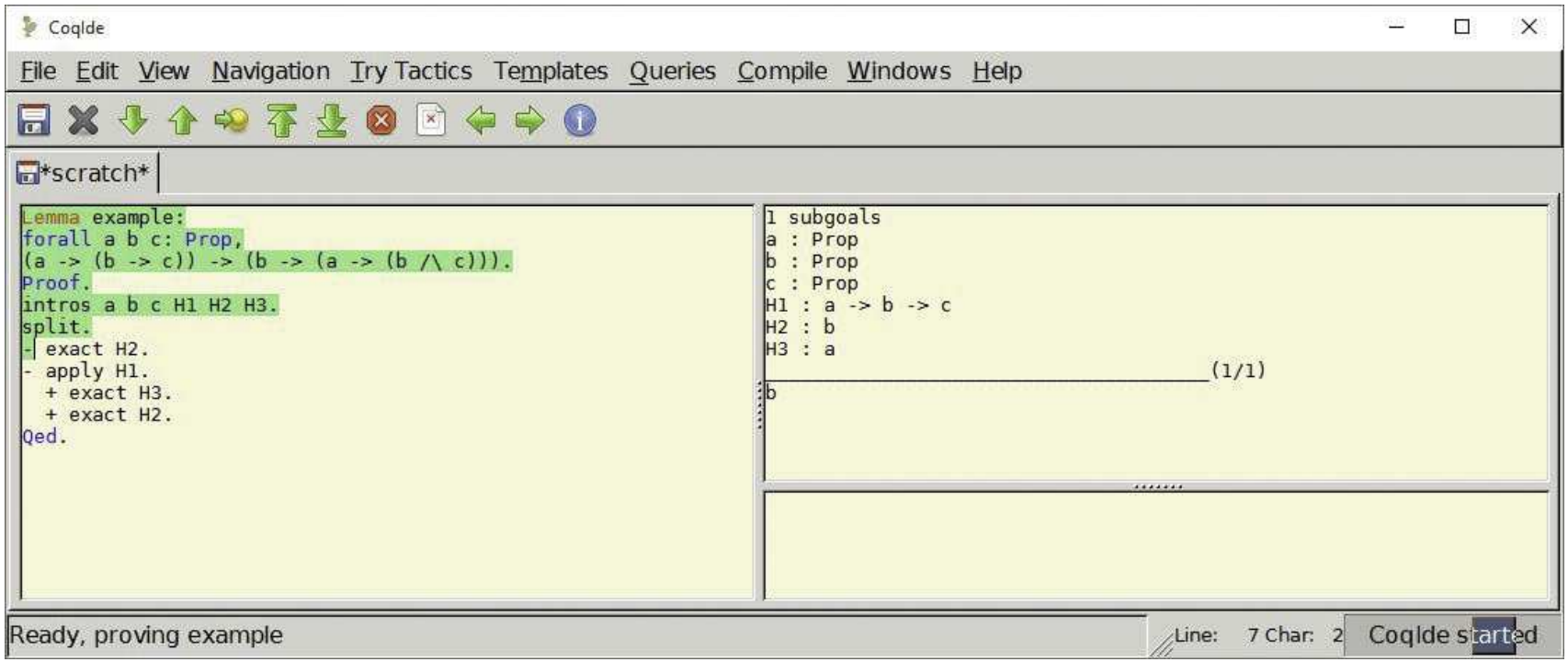
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
1 subgoals
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/1)
b /\ c
Ready, proving example
Line: 5 Char: 23
CoqIde started

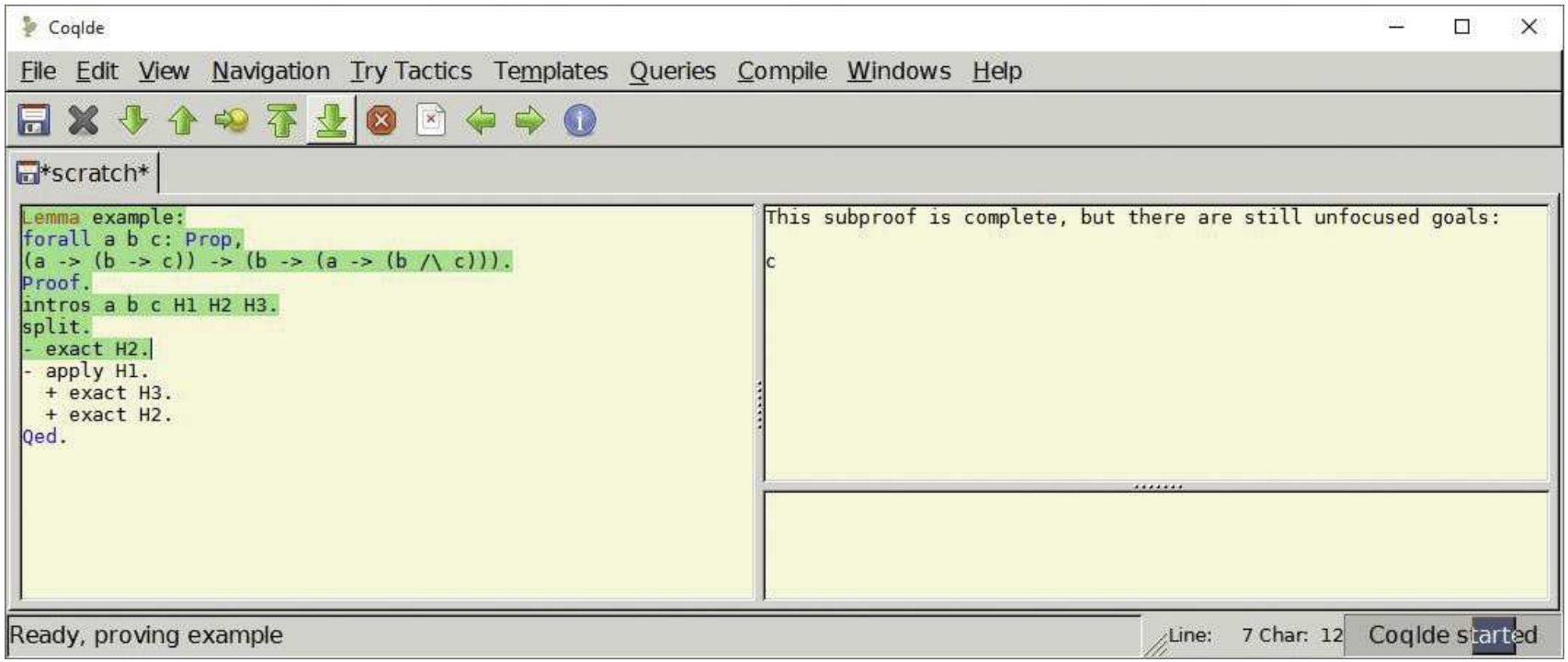
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
2 subgoal
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/2)
b
(2/2)
c
Ready, proving example
Line: 6 Char: 7
CoqIde started

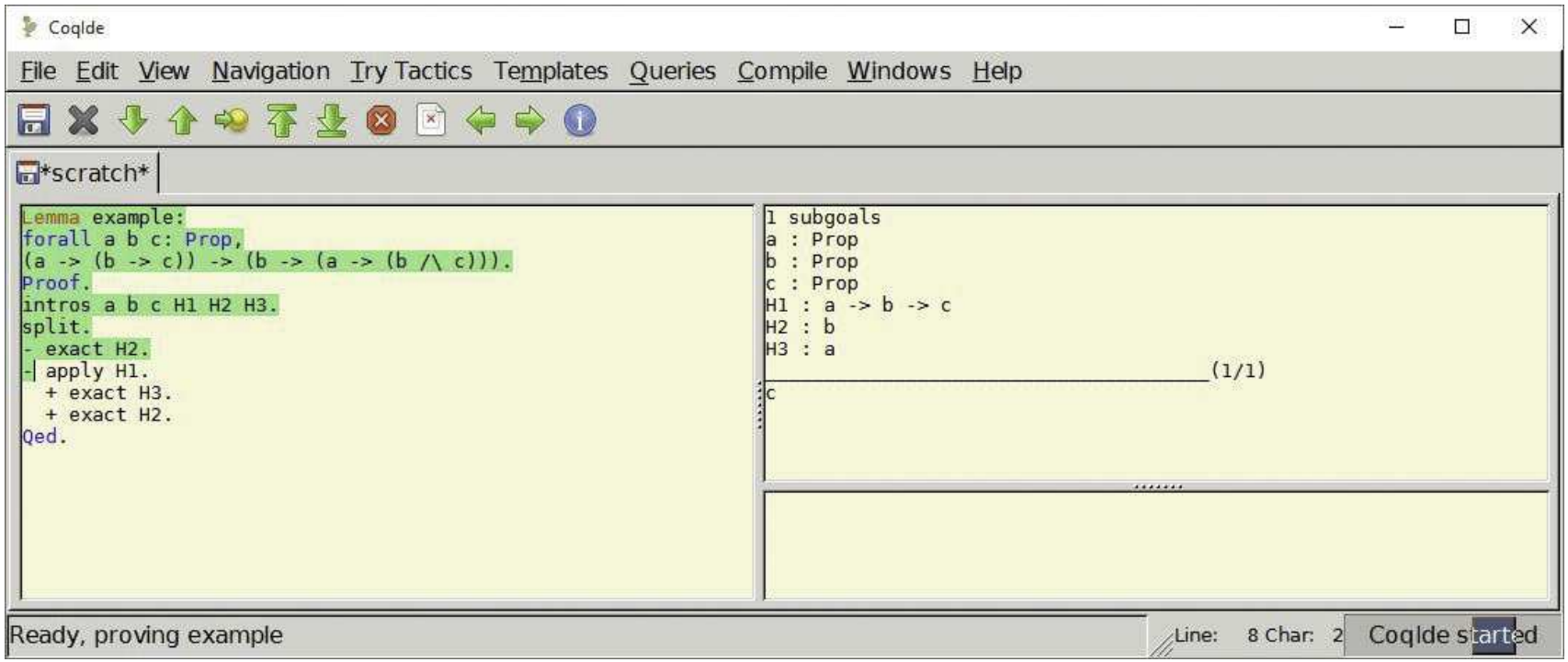
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
1 subgoals
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/1)
b
Ready, proving example
Line: 7 Char: 2
CoqIde started

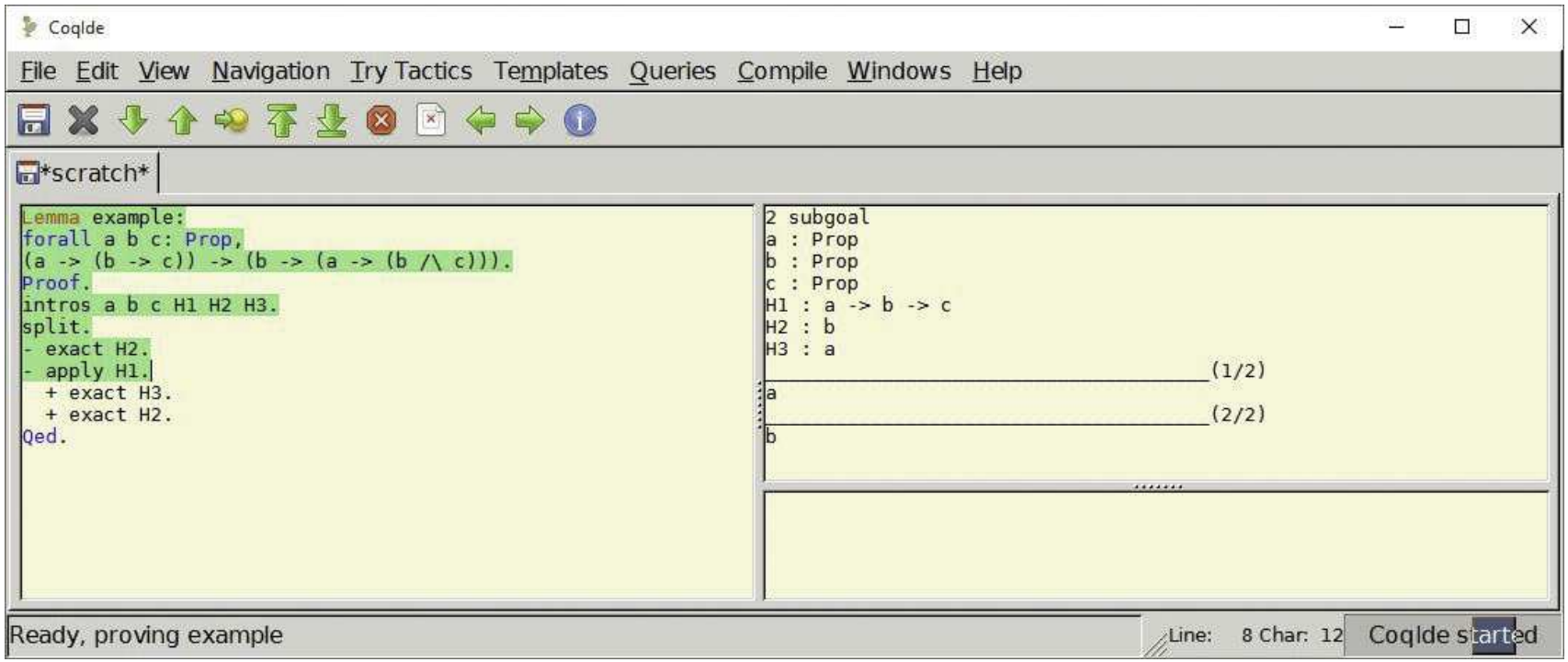
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
This subproof is complete, but there are still unfocused goals:
c
Ready, proving example
Line: 7 Char: 12
CoqIde started

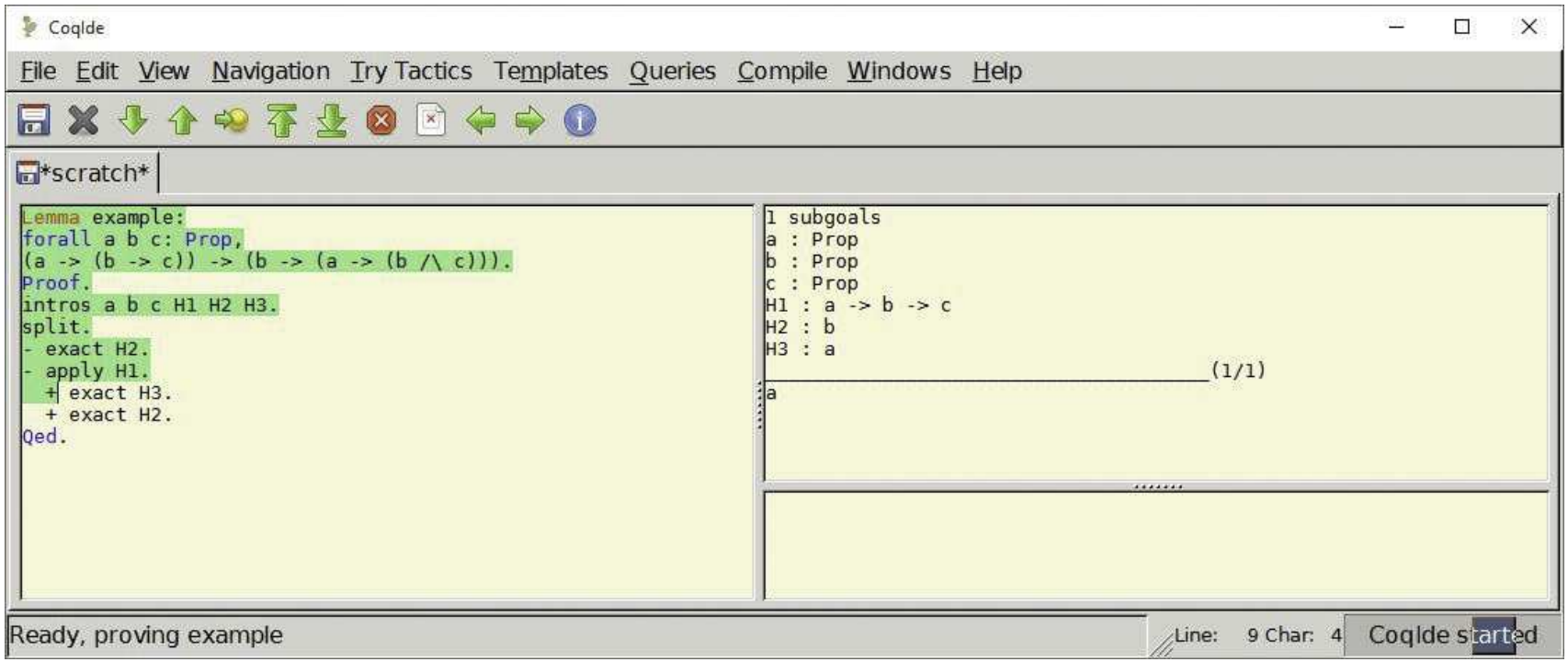
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
1 subgoals
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/1)
c
Ready, proving example
Line: 8 Char: 2
CoqIde started

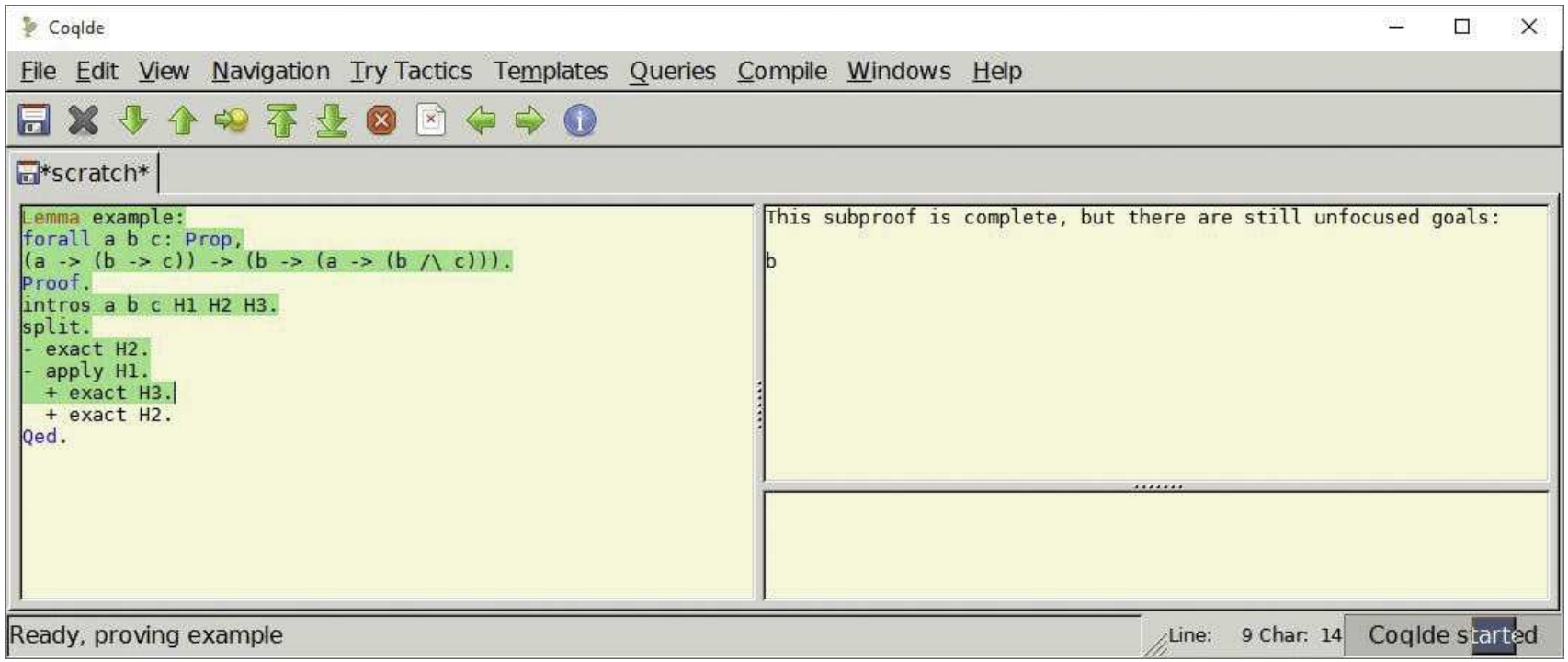
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
2 subgoal
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/2)
a
(2/2)
b
Ready, proving example
Line: 8 Char: 12
CoqIde started

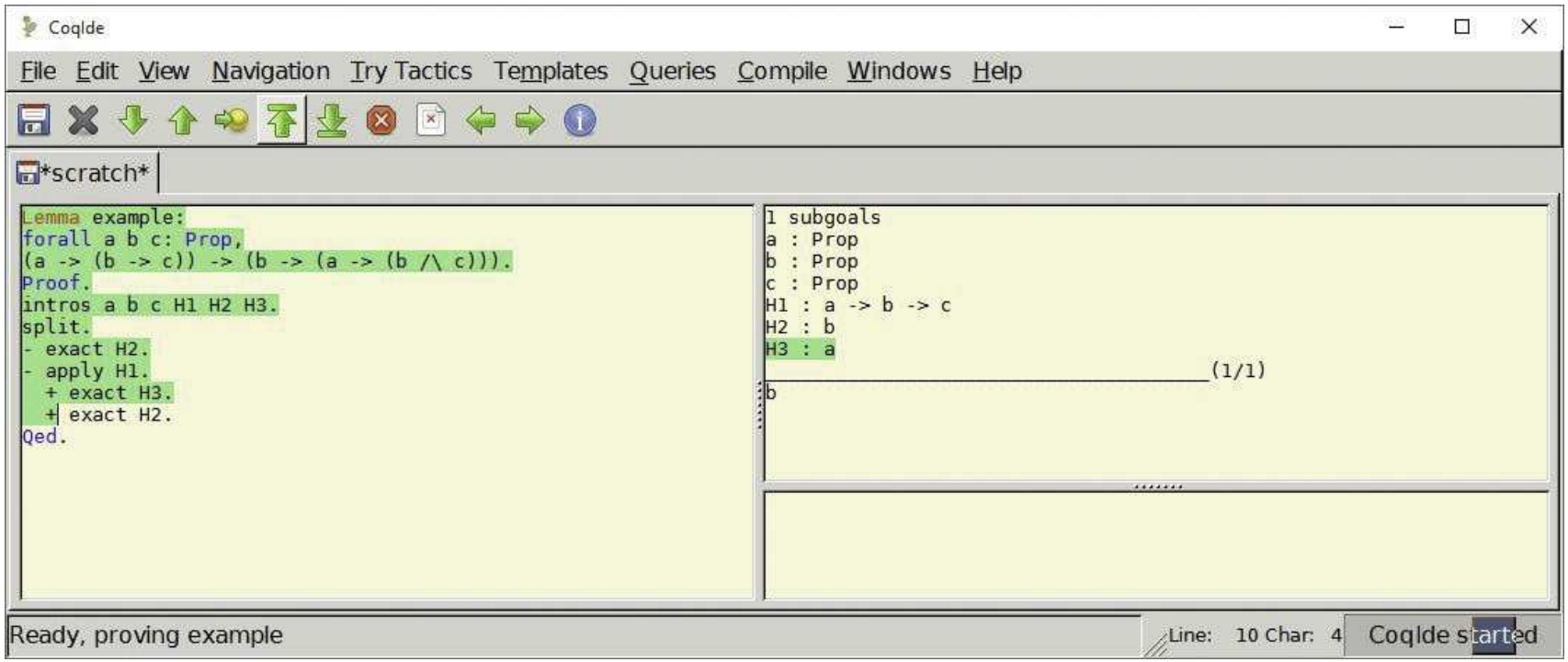
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
1 subgoals
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/1)
a
Ready, proving example
Line: 9 Char: 4
CoqIde started

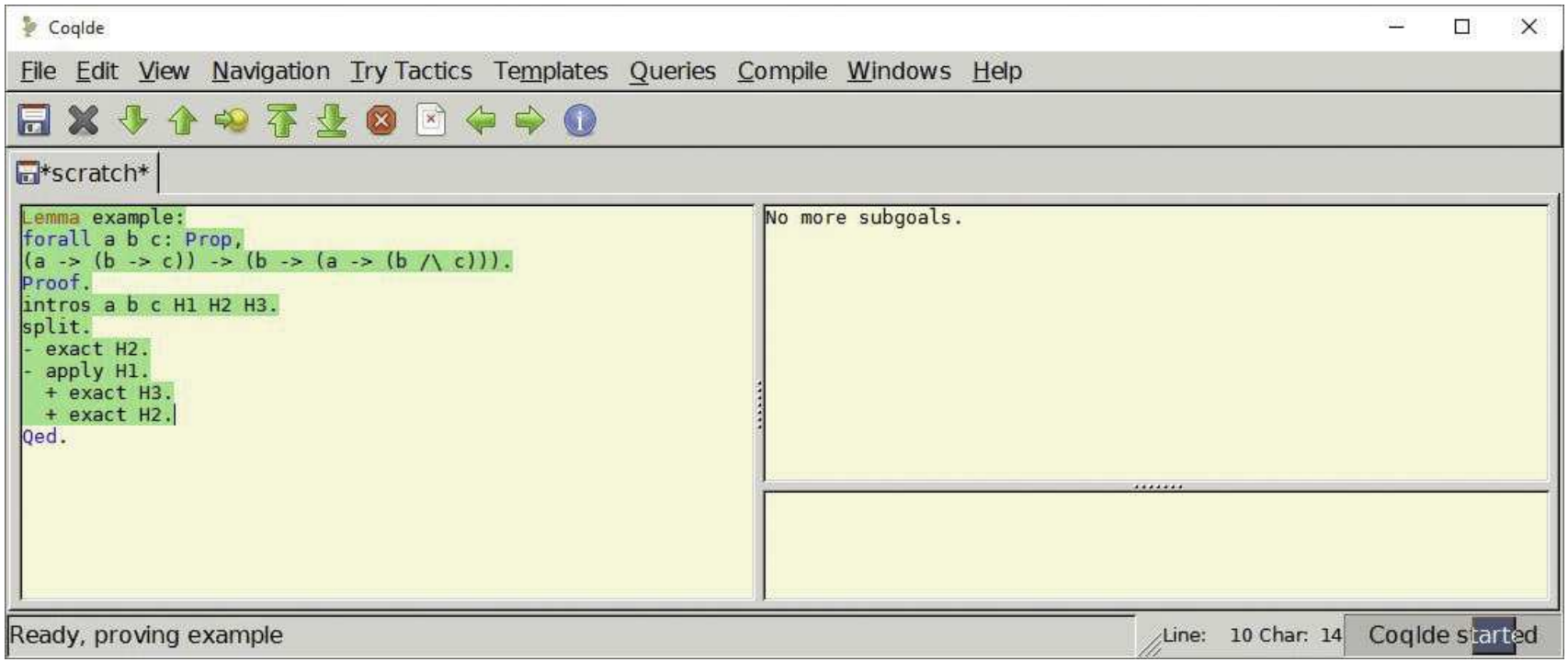
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
This subproof is complete, but there are still unfocused goals:
b
Ready, proving example
Line: 9 Char: 14
CoqIde started

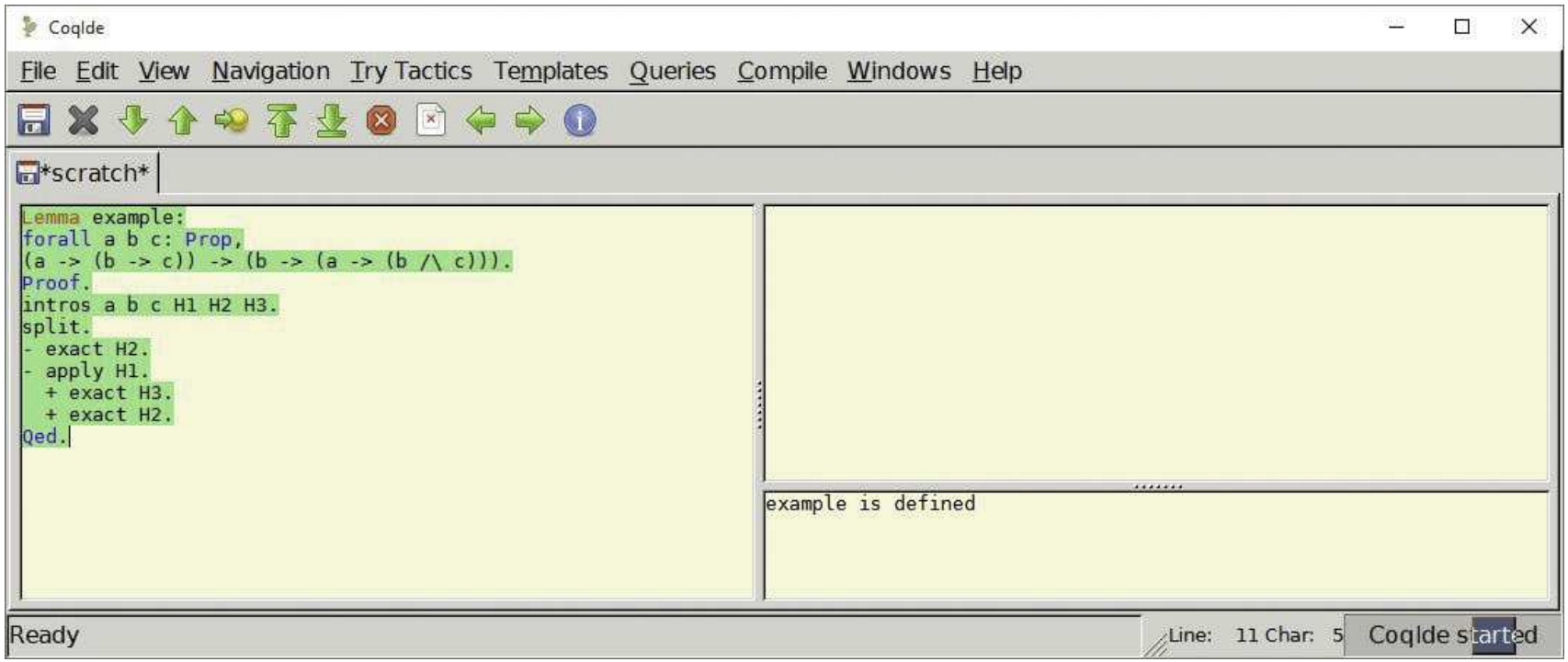
CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
1 subgoals
a : Prop
b : Prop
c : Prop
H1 : a -> b -> c
H2 : b
H3 : a
(1/1)
b
Ready, proving example
Line: 10 Char: 4
CoqIde started

CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
  + exact H3.
  + exact H2.
Qed.
No more subgoals.
Ready, proving example
Line: 10 Char: 14
CoqIde started

CoqIDE


CoqIde
File Edit View Navigation Try Tactics Templates Queries Compile Windows Help
scratch
Lemma example:
forall a b c: Prop,
(a -> (b -> c)) -> (b -> (a -> (b /\ c))).
Proof.
intros a b c H1 H2 H3.
split.
- exact H2.
- apply H1.
+ exact H3.
+ exact H2.
Qed.
example is defined
Ready
Line: 11 Char: 5
CoqIde started

Objetivos

Questões iniciais

Objetivos:

- Despertar o interesse pelo assunto;
- Apresentar uma técnica inovadora que está mudando a forma de se fazer matemática e de se desenvolver software;
- Estudar Provadores de Teoremas e Coq em particular;
- Entender o que é formalização matemática;
- Provar teoremas simples;
- Aprender como usar Coq para o desenvolvimento de software certificado;
- Incentivar o estudo continuado e a atuação na área, com pesquisas e publicações;
- Estimular a participação no nosso grupo de estudos.

Questões iniciais

Teorias envolvidas:

- Lógica;
- Teoria de Provas;
- Dedução Natural;
- Cálculo Lambda (não-tipado e tipado);
- Teoria de Tipos;
- Curry-Howard;
- Construtivismo;
- Técnicas de prova (indução etc)
- etc.

Questões iniciais

Objetos de estudo:

- Teorias (slide anterior);
- Coq;
- Exemplos;
- Exercícios;
- Estudos de caso;
- Slides, artigos e livros.

Muito estudo, muita persistência, muito tempo e muita dedicação.

Questões iniciais

Em resumo:

- Não é fácil;
- Aprendizado lento;
- Exige muita dedicação;
- Área ativa de pesquisa;
- Aplicações comerciais e acadêmicas de grande relevância;
- Muitas oportunidades na academia e na indústria;
- Tendência irreversível na matemática e na computação;
- É o futuro (da matemática e do desenvolvimento de software);
- Grande oportunidade.

Exemplo Completo

Especificação de um Programa

Aloritmo de ordenação

Como especificar um algoritmo de ordenação?

- Definir o domínio (listas de números inteiros);
- Relacionar entrada, saída e requisitos:
  - Entrada: uma lista de números inteiros (repetições são permitidas);
  - Saída: uma lista de números inteiros;
  - Requisito 1: as listas possuem os mesmos elementos (permutação);
  - Requisito 2: a lista de saída deve estar "ordenada".
- Escrever a proposição/especificação;
- Provar o teorema/construir o programa que implementa a especificação.

Objetivo

- Construir um programa certificado que ordena uma lista de números inteiros;
- Passos:
  - Formular a especificação do programa na forma de uma proposição da lógica de predicados;
  - Tratar a especificação como um teorema e construir a prova do mesmo;
  - Extrair o programa certificado a partir da prova.
- Extraído do livro:
  Interactive Theorem Proving and Program Development
  Yves Bertot e Pierre Castéran

Observações gerais

- Muitos detalhes;
- Não se preocupem em entender tudo;
- Busquem apenas uma intuição inicial do que está sendo feito e como está sendo feito;
- Mais importante é ter uma visão geral da dinâmica e do tipo de trabalho envolvido;
- A plena compreensão virá depois, com o tempo e a prática.

Script Coq

- Texto corrido;
- Processado de cima para baixo, esquerda para a direita;
- Mensagens de erros e interação com o usuário;
- Definições (indutivas e não-indutivas);
- Funções (recursivas e não-recursivas);
- Lemas e teoremas (proposições provadas de forma interativa usando um conjunto de táticas e regras de inferência; as provas são criadas indiretamente).

Script Coq

- Novos nomes são introduzidos em cada nova definição, lema, teorema ou outro;
- Utilização nas etapas seguintes;
- (lema A é usado para provar B, que por sua vez é usado para provar C e assim por diante)
- Computação e dedução;
- Lema ou teorema final;
- Provas completas;
  - Contexto;
  - Indução;
  - O script não é a prova!
- Extração de código.

Objetivo

Construir um programa certificado que ordena listas de números inteiros.

- Número inteiro?
- Lista?
- Lista ordenada?
- Qual seria a especificação deste programa?
- Uma vez especificado, como construímos a prova?
- Da prova, como extraímos o programa certificado?
- Série de definições (algumas indutivas outras não) e lemas.

Número Natural

- Um tipo de dados definido de maneira indutiva:
  - Existe pelo menos um caso base;
  - Existe pelo menos um caso indutivo.
- Dois construtores apenas;
- Construtores são funções usadas para construir os valores do tipo que está sendo definido;
- A expressão à direita do " : " representa o tipo da função;
- O construtor (função) `O` não tem argumentos e representa o valor "zero" (caso base);
- O construtor (função) `S` tem um único argumento e representa "sucessor" de um número natural, que também é um número natural (caso indutivo);
- Os números naturais são representados em unário.

Número Natural

Definição de número natural em Coq:

```
Inductive nat: Type :=
| O : nat
| S : nat → nat.
```

Exemplos:

O	(0)
S O	(1)
S (S O)	(2)
S (S (S O))	(3)
...	(...)

Lista

Definição de lista em Coq:

- Um tipo de dados definido de maneira indutiva:
  - Existe pelo menos um caso base;
  - Existe pelo menos um caso indutivo.
- Parametrizado em função do tipo do elemento (A);
- Dois construtores apenas:
  - `nil` representa "lista vazia" (caso base);
  - `cons` representa "acréscimo de elemento à esquerdasucessor" (caso indutivo).
- Tipo polimórfico (serve para qualquer tipo de elemento);
- Faz uso intensivo de "notações".

Lista

Definição de lista em Coq:

```
Inductive list (A : Type) : Type :=
| nil : list A
| cons : A → list A → list A.
```

Exemplos:

nil	nil	[]
cons 3 nil	3 :: nil	[3]
cons (3 (cons (4 (cons 5 nil))))	3 :: 4 :: 5 :: nil	[3;4;5]

Lista Ordenada

Definição de lista ordenada em Coq:

- Uma proposição definida de maneira indutiva:
  - Existe pelo menos um caso base;
  - Existe pelo menos um caso indutivo.
- Uma coleção infinita de proposições definida de maneira indutiva;
- Usaremos números inteiros (`Z`) no lugar de números naturais (`nat`).
- Três construtores:
  - `sorted0`: lista vazia é ordenada por definição;
  - `sorted1`: lista com um único elemento é ordenada por definição;
  - `sorted2`: um número menor ou igual que o cabeça de uma lista ordenada, quando inserido no início da mesma, produz uma lista igualmente ordenada;
- Os construtores de uma proposição definida de maneira indutiva são considerados axiomas, proposições que são aceitas válidas sem provas.

Lista Ordenada

Definição de lista ordenada em Coq:

```
Inductive sorted : list Z → Prop :=
| sorted0 : sorted nil
| sorted1 : ∀ z:Z, sorted (z::nil)
| sorted2 : ∀ z1 z2:Z,
            ∀ l:list Z,
            z1 ≤ z2 → sorted (z2::l) → sorted (z1::z2::l).
```

Exemplos:

Proposição	Construtores utilizados
`sorted nil`	`sorted0`
`sorted (3::nil)`	`sorted1`
`sorted (2::3::nil)`	`sorted1` e `sorted2`
`sorted (1::2::3::nil)`	`sorted1`, `sorted2` e `sorted2`

Prova de Ordenação

Prova de que a lista `2::3::5::7::nil` é ordenada:

```
Lemma sorted_example:
sorted (2::3::5::7::  nil).
Proof.
apply sorted2.
omega.
apply sorted2.
  + omega.
  + apply sorted2.
    * omega.
    * apply sorted1.
Qed.
```

Sublista Ordenada

Teorema auxiliar que prova que a remoção do elemento cabeça de uma lista ordenada preserva a ordenação da lista restante:

```
Theorem sorted_inv :
∀ z:Z,
∀ l:list Z,
sorted (z::l) → sorted l.
Proof.
intros z l H.
inversion H.
apply sorted0.
exact H3.
Qed.
```

Número de Ocorrências

Função recursiva que computa o número de ocorrências de um mesmo elemento numa lista:

```
Fixpoint nb_occ (z:Z) (l:list Z): nat:=
match l with
| nil ⇒ 0
| (z' :: l') ⇒
  match Z_eq_dec z z' with
  | left _ ⇒ S (nb_occ z l')
  | right _ ⇒ nb_occ z l'
  end
end.
```

Permutação

Proposição (predicado) que indica se uma lista é ou não é ordenada:

```
Definition permutation (l l':list Z) : Prop :=
∀ z:Z,
nb_occ z l = nb_occ z l'.
```

A palavra-chave "`Definition`" também é usada para introduzir funções não-recursivas. Se a função for recursiva deve-se usar "`Fixpoint`".

Inserção

Função recursiva que insere um número inteiro numa lista ordenada, de modo que a mesma continue ordenada:

```
Fixpoint insert (z:Z) (l:list Z): list Z :=
match l with
| nil ⇒ z :: nil
| cons a l' ⇒
  match Z_le_gt_dec z a with
  | left _ ⇒ z ::  a ::  l'
  | right _ ⇒ a :: (insert z l')
  end
end.
```

Em Resumo

Temos todos os elementos para formular a proposição que se deseja provar:

- Sabemos os que é um número natural (e inteiro);
- Sabemos o que é uma lista;
- Sabemos o que é uma lista ordenada;
- Sabemos o que é uma permutação;
- Sabemos inserir numa lista preservando a ordenação.

Portanto, podemos formular a especificação que desejamos provar.

Objetivo

Provar a proposição:

```
Lemma sort_correct:
∀ l: list Z,
∃ l': list Z,
permutation l l' ∧ sorted l'.
```

- A prova desta proposição garante a existência de uma lista ordenada equivalente (com os mesmos elementos) para qualquer outra que se considere;
- Um programa certificado pode ser extraído desta prova.

Script Coq da Prova

```
Proof.
induction l.
- ∃ nil.
 split.
 + apply permutation_refl.
 + apply sorted0.
- destruct IHl as [l' [H1 H2]].
 ∃ (insert a l').
 split.
 + apply permutation_trans with (l2:= a :: l').
  * apply permutation_cons.
   exact H1.
  * apply insert_permutation.
 + apply insert_sorted.
  exact H2.
Qed.
```

A Prova

```
sort_correct =
fun l : list Z =>
list_ind
  (fun l0 : list Z => exists l' : list Z, permutation l0 l' /\ sorted l')
  (ex_intro (fun l' : list Z => permutation nil l' /\ sorted l') nil
     (conj (permutation_refl nil) sorted0))
  (fun (a : Z) (l0 : list Z)
     (IHl : exists l' : list Z, permutation l0 l' /\ sorted l') =>
   match IHl with
   | ex_intro _ l' (conj H1 H2) =>
       ex_intro (fun l'0 : list Z => permutation (a :: l0) l'0 /\ sorted l'0)
         (insert a l')
         (conj
            (permutation_trans (a :: l0) (a :: l') (insert a l')
               (permutation_cons a l0 l' H1) (insert_permutation l' a))
            (insert_sorted l' a H2))
   end) l
     : forall l : list Z, exists l' : list Z, permutation l l' /\ sorted l'
```

O Programa Extraído 1(4)

```
type __ = Obj.t
let __ = let rec f _ = Obj.repr f in Obj.repr f
type 'a list =
| Nil
| Cons of 'a * 'a list
type comparison =
| Eq
| Lt
| Gt
(** val compOpp : comparison -> comparison **)
let compOpp = function
| Eq -> Eq
| Lt -> Gt
| Gt -> Lt
type sumbool =
| Left
| Right
type positive =
| XI of positive
| XO of positive
| XH
type z =
| Z0
| Zpos of positive
| Zneg of positive
```

O Programa Extraído 2(4)

```
module Pos =
 struct
  (** val compare_cont : comparison -> positive -> positive -> comparison **)
  let rec compare_cont r x y =
    match x with
    | XI p ->
      (match y with
       | XI q -> compare_cont r p q
       | XO q -> compare_cont Gt p q
       | XH -> Gt)
    | XO p ->
      (match y with
       | XI q -> compare_cont Lt p q
       | XO q -> compare_cont r p q
       | XH -> Gt)
    | XH ->
      (match y with
       | XH -> r
       | _ -> Lt)
  (** val compare : positive -> positive -> comparison **)
  let compare =
    compare_cont Eq
 end
```

O Programa Extraído 3(4)

```
module Z =
 struct
  (** val compare : z -> z -> comparison **)
  let compare x y =
    match x with
    | Z0 ->
      (match y with
       | Z0 -> Eq
       | Zpos _ -> Lt
       | Zneg _ -> Gt)
    | Zpos x' ->
      (match y with
       | Zpos y' -> Pos.compare x' y'
       | _ -> Gt)
    | Zneg x' ->
      (match y with
       | Zneg y' -> compOpp (Pos.compare x' y')
       | _ -> Lt)
 end
```

O Programa Extraído 4(4)

```
(** val z_le_dec : z -> z -> sumbool **)
let z_le_dec x y =
  match Z.compare x y with
  | Gt -> Right
  | _ -> Left
(** val z_le_gt_dec : z -> z -> sumbool **)
let z_le_gt_dec x y =
  z_le_dec x y
(** val insert : z -> z list -> z list **)
let rec insert z0 = function
| Nil -> Cons (z0, Nil)
| Cons (a, l') ->
  (match z_le_gt_dec z0 a with
   | Left -> Cons (z0, (Cons (a, l')))
   | Right -> Cons (a, (insert z0 l')))
(** val sort_correct : __ **)
let sort_correct =
  __
```

ProofWeb

ProofWeb

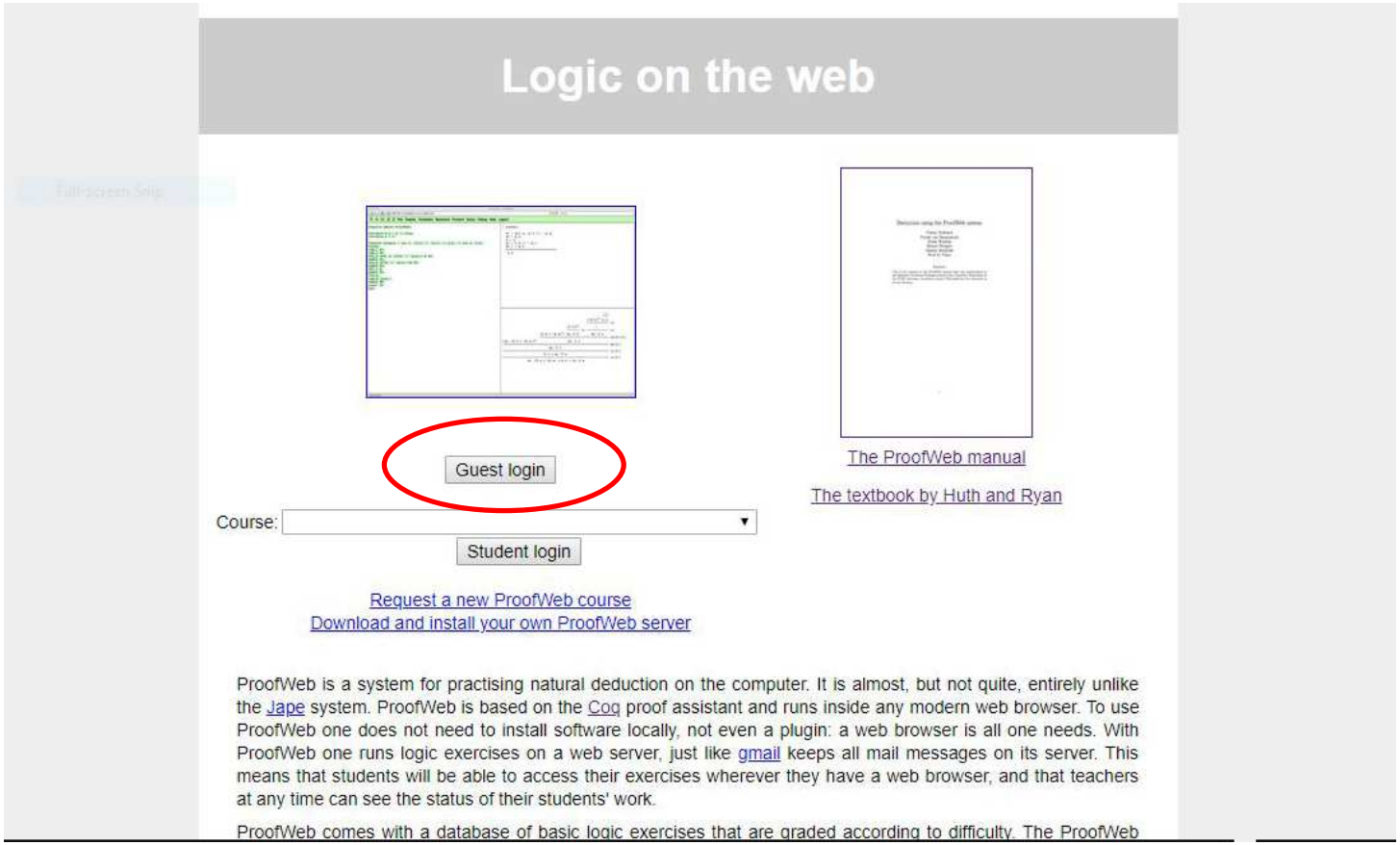
Versão web do Coq:

- Pode ser usada via navegador (Firefox);
- Não precisa baixar nem instalar;
- Disponível em `http://proofweb.cs.ru.nl`;
- Clicar em "Guest login";
- Clicar em "Access the interface";
- Alternativamente, é possível se identificar e salvar os arquivos;
- Oferece também cursos na área;
- Suporta diversos assistentes de prova.

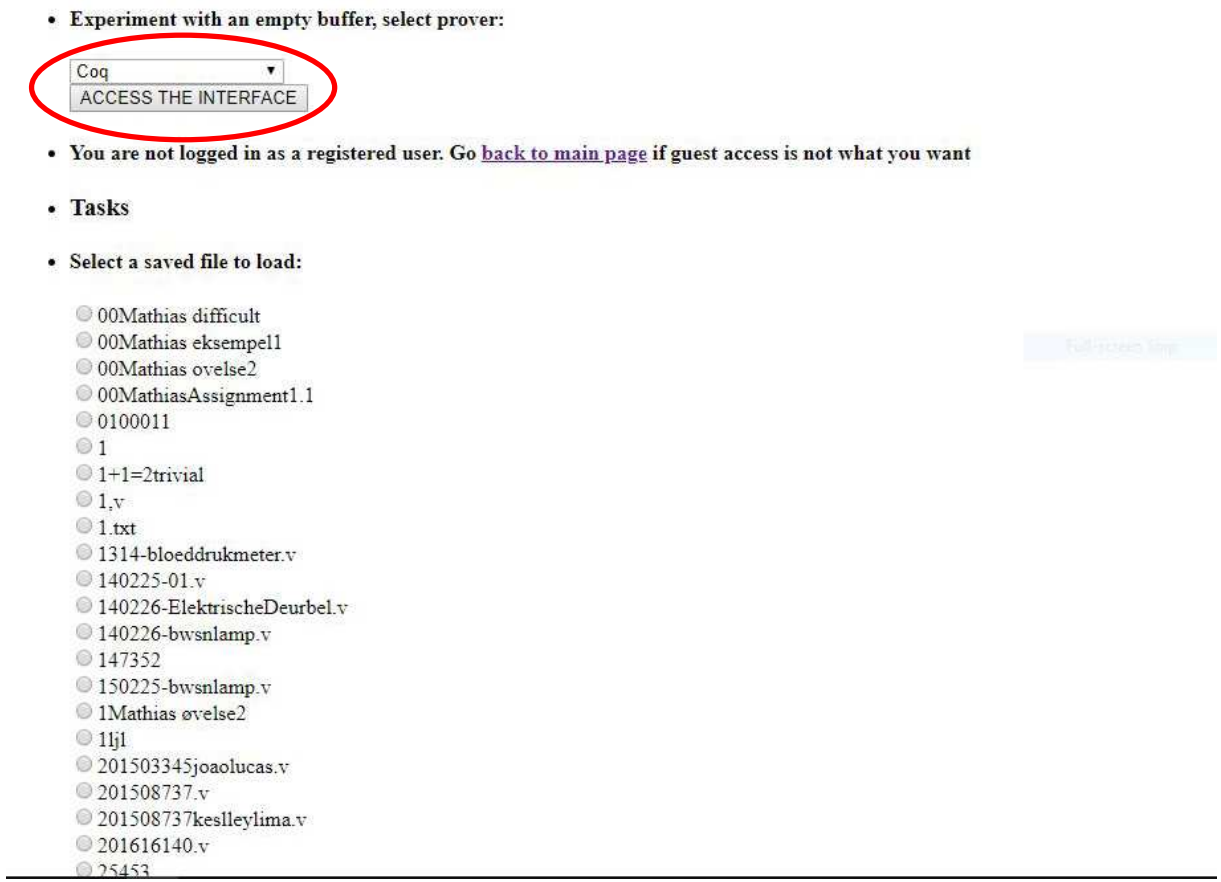
ProofWeb 1(4)


Courses
Provers
MathWiki
Calculator
ProofWeb
SURF FOUNDATION
What is ProofWeb?
ProofWeb is both a system for teaching logic and for using proof assistants through the web.
ProofWeb can be used in three ways. First, one can use the guest login, for which one does not even need to register. Secondly, a user can be a student in a logic or proof assistants course. We are hosting courses free of charge. If you are a teacher and would like to host your course on this server, send email to proofweb@cs.ru.nl. Thirdly, if teachers do not want to trust us with their students' files, they can freely download the ProofWeb system and run it on a server of their own.
ProofWeb works well with many web browsers, but it does not work with all versions of Internet Explorer. ProofWeb was developed using the Firefox browser, which can be downloaded for free.

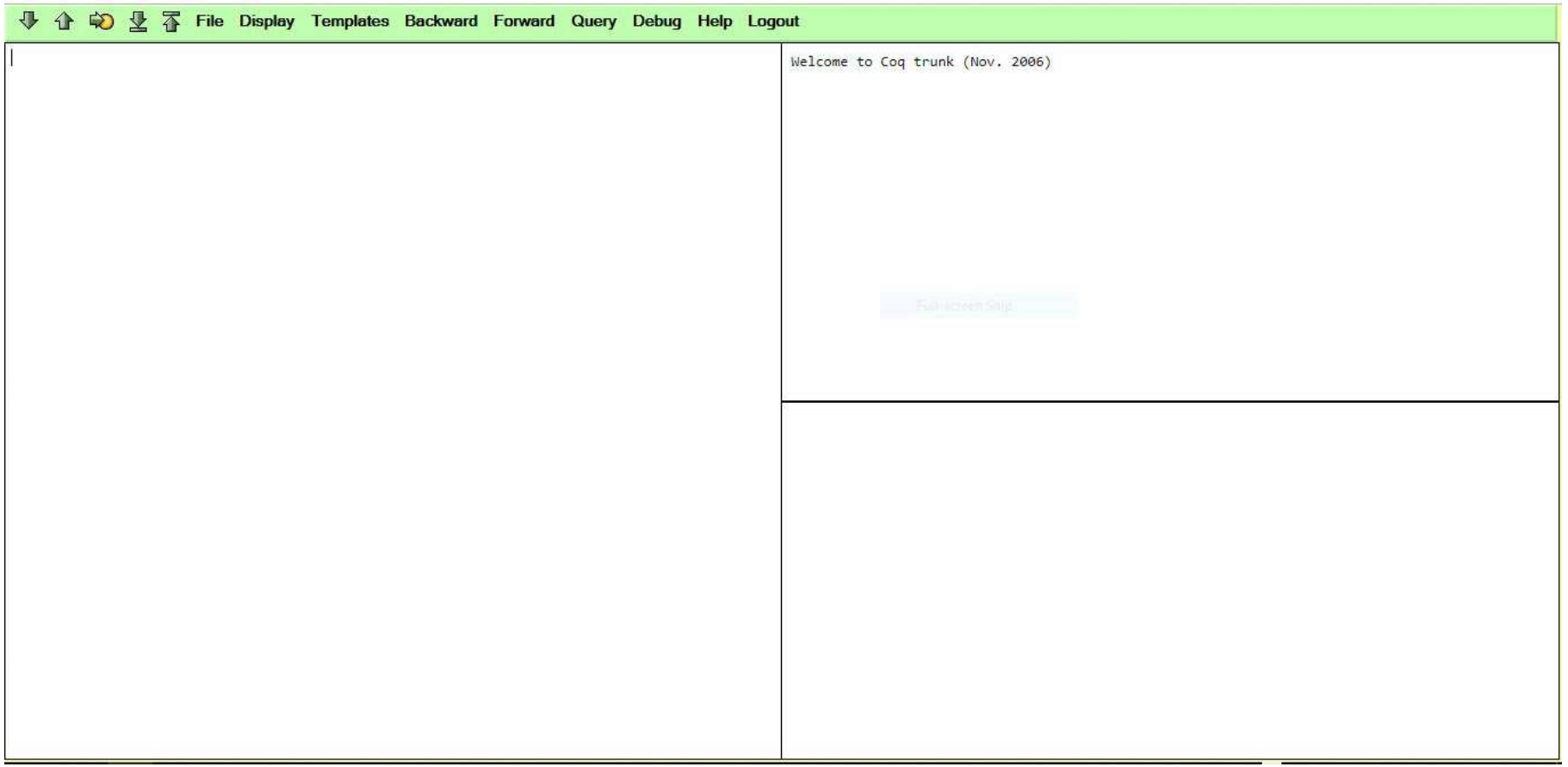
ProofWeb 2(4)


Logic on the web
Guest login
Course:
Student login
Request a new ProofWeb course
Download and install your own ProofWeb server
The ProofWeb manual
The textbook by Huth and Ryan
ProofWeb is a system for practising natural deduction on the computer. It is almost, but not quite, entirely unlike the Jape system. ProofWeb is based on the Coq proof assistant and runs inside any modern web browser. To use ProofWeb one does not need to install software locally, not even a plugin: a web browser is all one needs. With ProofWeb one runs logic exercises on a web server, just like gmail keeps all mail messages on its server. This means that students will be able to access their exercises wherever they have a web browser, and that teachers at any time can see the status of their students' work.
ProofWeb comes with a database of basic logic exercises that are graded according to difficulty. The ProofWeb

ProofWeb 3(4)


Experiment with an empty buffer, select prover:
Coq
ACCESS THE INTERFACE
You are not logged in as a registered user. Go back to main page if guest access is not what you want
Tasks
Select a saved file to load:
00Mathias difficult
00Mathias eksempel1
00Mathias ovelse2
00MathiasAssignment1.1
0100011
1
1+1=2trivial
1,v
1.txt
1314-bloeddrukmeter.v
140225-01.v
140226-ElektrischeDeurbel.v
140226-bwsnlamp.v
147352
150225-bwsnlamp.v
1Mathias øvelse2
1ljl
201503345joaolucas.v
201508737.v
201508737keslleylima.v
201616140.v
25453

ProofWeb 4(4)


File
Display
Templates
Backward
Forward
Query
Debug
Help
Logout
Welcome to Coq trunk (Nov. 2006)

Observações ao utilizar o ProofWeb 1(2)

- Bullets `-`, `+`, `*` não são aceitos;
- O comando "`Compute`" deve ser substituído por "`Eval red in`" ou "`Eval vm_compute in`":

```
Compute (next_weekday friday).
Eval red in (next_weekday friday).
```

Observações ao utilizar o ProofWeb 2(2)

- O argumento descrecente deve ser explicitado nas funções recursivas com mais de um argumento:

```
Fixpoint plus (n : nat) (m : nat) : nat :=
match n with
| O ⇒ m
| S n' ⇒ S (plus n' m)
end.
Fixpoint plus (n : nat) (m : nat) {struct n}: nat :=
match n with
| O ⇒ m
| S n' ⇒ S (plus n' m)
end.
```

Teoria

Introduction

Formalização Matemática

- Construção de provas assistida por máquina;
- Verificação mecanizada de provas;
- Velocidade, confiabilidade e reutilização;
- Matemática e Ciência da Computação;
- Prova interativa de teoremas;
- Desenvolvimento certificado de hardware e software.

Casos reais

Formalização matemática é uma atividade madura:

- Usada ao longo dos anos;
- Diversidade de assistentes de provas e teorias subjacentes;
- Desenvolvimento da tecnologia dos assistentes de provas;
- Tamanho, complexidade e importância de diferentes projetos;
- Orientação teórica e tecnológica;
- Indústria e academia;
- Uma tendência clara;
- Ponto sem volta.

Quadro Geral

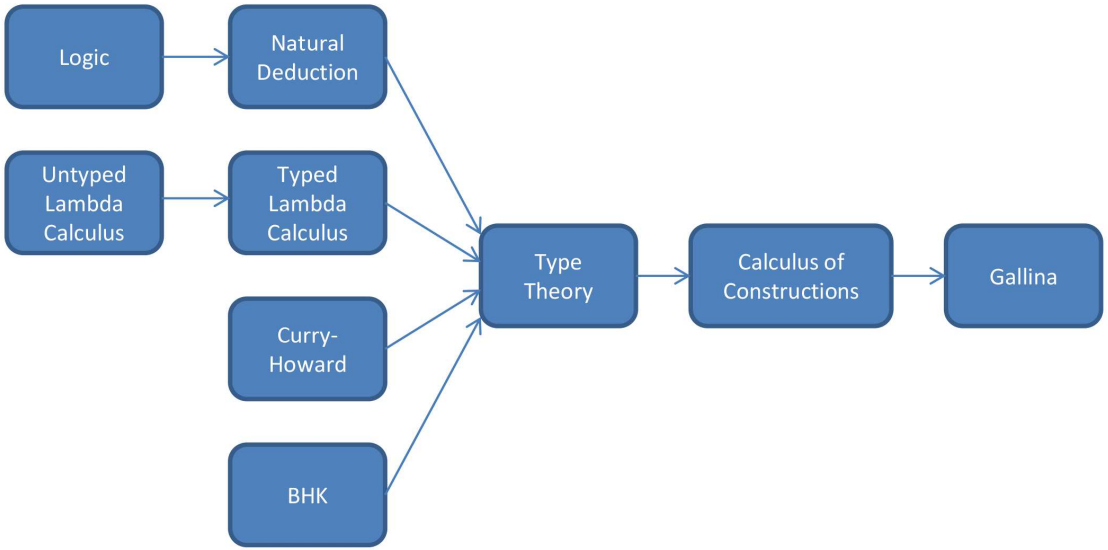
- Matemática "informal":
  - Diferentes níveis de abstração podem esconder erros difíceis de serem identificados;
  - Notação não-uniforme também pode constituir um problema.
- Formalização matemática (*"matemática codificada no computador"*) é uma tendência clara na direção do desenvolvimento teórico e da representação da teoria;
- Raciocínio auxiliado por computador e o uso de assistentes interativos de prova;
- Verificação mecânica de provas e programas, permitindo:
  - Verificação de cada passo de inferência contra um conjunto de regras de inferência da lógica subjacente;
  - Notação uniforme.
- Vantagens:
  - Menos esforço e tempo;
  - Maior confiabilidade.

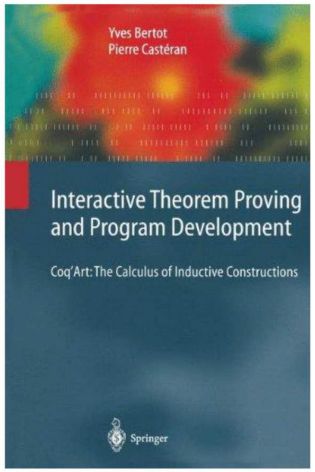
Requisitos

Requisitos teóricos para usar e entender o Coq:

- Lógica;
- Dedução Natural;
- Cálculo Lambda Não-Tipado;
- Cálculo Lambda Tipado;
- Correspondência de Curry-Howard;
- Teoria de Tipos;
- Construtivismo e BHK;
- Teoria de Tipos Intuicionística de Martin Löf;
- Cálculo de Construções com Definições Indutivas.

Background


Logic
Natural Deduction
Untyped Lambda Calculus
Typed Lambda Calculus
Curry-Howard
BHK
Type Theory
Calculus of Constructions
Gallina


Yves Bertot
Pierre Castéran
Interactive Theorem Proving and Program Development
Coq'Art: The Calculus of Inductive Constructions
Springer

Referências

Referências

Estão todas disponíveis na página do nosso grupo de estudos:
"Provadores de Teoremas e suas Aplicações"
`http://marcusramos.com.br/univasf/provadores/`

- Artigos;
- Livros;
- Slides;
- Links;
- Exercícios com solução;
- e muito mais.

Recomendações especiais

- Introdução ao Coq:
  Software Foundations Vol. 1 - Logical Foundations
  (Pierce et al)
- Coq:
  Interactive Theorem Proving and Program Development
  (Bertot & Castéran)
- Teoria do Coq (Cálculo de Construções):
  Type Theory and Formal Proof
  (Nederpelt & Geuvers)

Conclusões

Fato

Provadores de Teoremas são o futuro (e o presente):

- Da matemática;
- Do desenvolvimento de software.

Tanto na industria quanto na academia.

Matemática

Principais características do processo:

- Verificação mecânica de provas;
- Assistência na construção de provas;
- Reaproveitamento de scripts;
- Repositórios;

Principais benefícios derivados:

- Produtividade;
- Correção;
- Uniformidade;
- Agilidade na publicação de originais.

Desenvolvimento de Software Certificado

Roteiro básico:

1. Escreva as especificações como expressões de tipo;
2. Use uma lógica poderosa o suficiente e certifique-se de que a especificação esteja correta;
3. Interprete a especificação como um teorema;
4. Construa a prova do teorema usando uma lógica construtiva;
5. Use o provador de teoremas para verificar a prova;
6. Converta a prova para um programa de computador usando o recurso de extraçao de código.

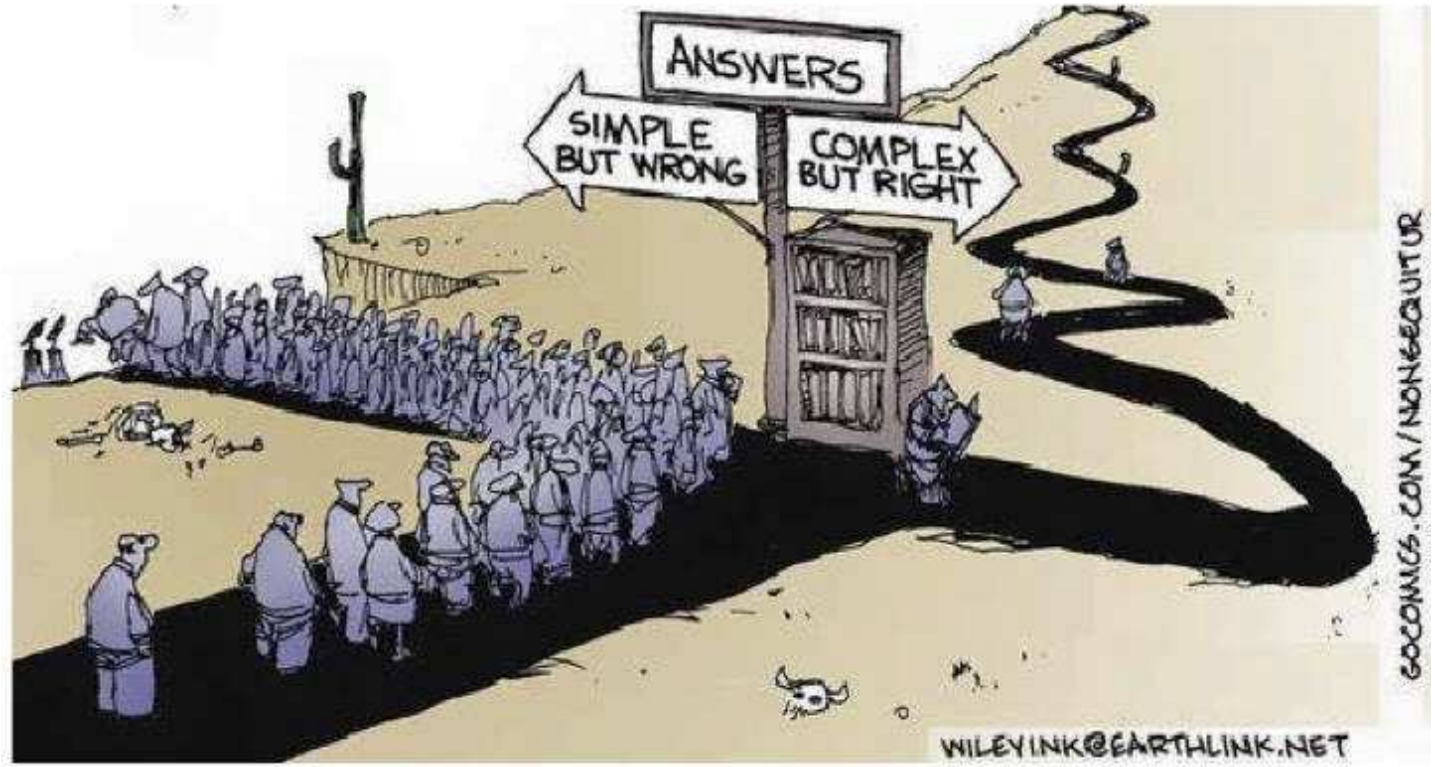
Uso de métodos matemáticos no lugar de métodos empíricos e subjetivos.

Certificação de Software já Desenvolvido

Roteiro básico:

1. Construir um termo que represente o programa;
2. Obter a expressão de tipo deste termo;
3. Verificar se a mesma corresponde à especificação desejada.

Computadores e Matemática



Computadores e Matemática

- Não é fácil mas é muito recompensador;
- Espero que vocês tenham gostado;
- Me perguntem se quiserem mais referências;
- Me escrevam se tiverem perguntas ou sugestões;
- Me avisem caso planejem trabalhar nesta área.

Obrigado!