Traf: a Graphical Proof Tree Viewer Cooperating with Coq through Proof General

Hideyuki Kawabata
Yuta Tanaka
Mai Kimura
Tetsuo Hironaka

Hiroshima City University

APLAS 2018, Wellington

Readability of (tactic-based/procedural) formal proofs

Formal vs. Informal Proof --- Software Foundations

"A proof is an act of communication."

"Formal proofs are useful in many ways, but they are not very efficient ways of communicating ideas between human beings."

(You have to) "step through the tactics one after the other in your mind".

due to the lack of context and goal stack at each point

How about writing contexts and/or subgoals in the proof?

Styles of formal proofs (and tools)

Declarative proofs	Tactic-based / procedural
😄 informative, readable without tools	😣 hard to read the proof scripts
😣 laborious to write intermediate formulae	😄 preferable for writing concise proofs interactively by making use of the theorem prover's automation facilities

(There are many combined approaches ...)

Do we have to change the style of formal proof?

Our approach:

graphically

provide support for writing and reading proof scripts

"interactive monitor"

"proof script browser"

for Proof General users

Visualizing a proof as a proof tree

```
Theorem pq_qp: forall P Q: Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q.
  intros H.
  destruct H as [HP | HQ].
  right. assumption.
  left. assumption.
Qed.
```

$$\frac{A}{A \vee B}\ (\vee\text{-introl})$$

$$\frac{A[y/x]}{\forall x.A}\ (\forall\text{-intro})$$

$$\frac{B}{A \vee B}\ (\vee\text{-intror})$$

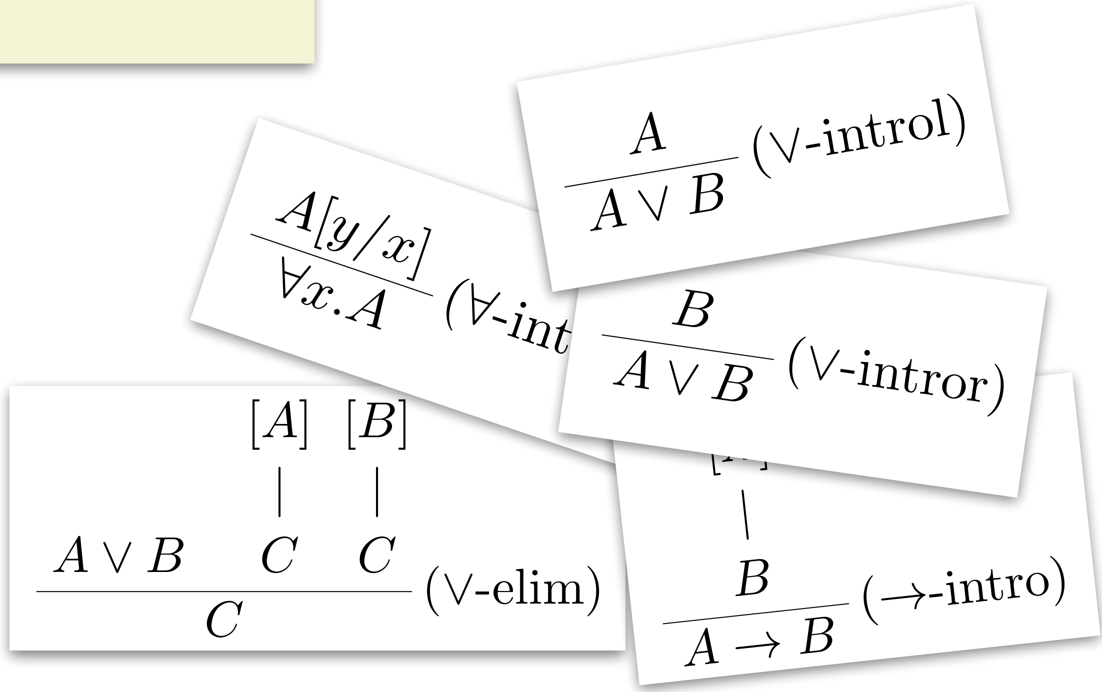
$$\frac{A \vee B \quad \begin{array}{c}[A]\\ \vert\\ C\end{array} \quad \begin{array}{c}[B]\\ \vert\\ C\end{array}}{C}\ (\vee\text{-elim})$$

$$\frac{\begin{array}{c}[A]\\ \vert\\ B\end{array}}{A \rightarrow B}\ (\rightarrow\text{-intro})$$

Gentzen-style natural deduction proof

Visualizing a proof as a proof tree

```
Theorem pq_qp: forall P Q: Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q.
  intros H.
  destruct H as [HP | HQ].
  right. assumption.
  left. assumption.
Qed.
```



Gentzen-style natural deduction proof

Visualizing a proof as a proof tree

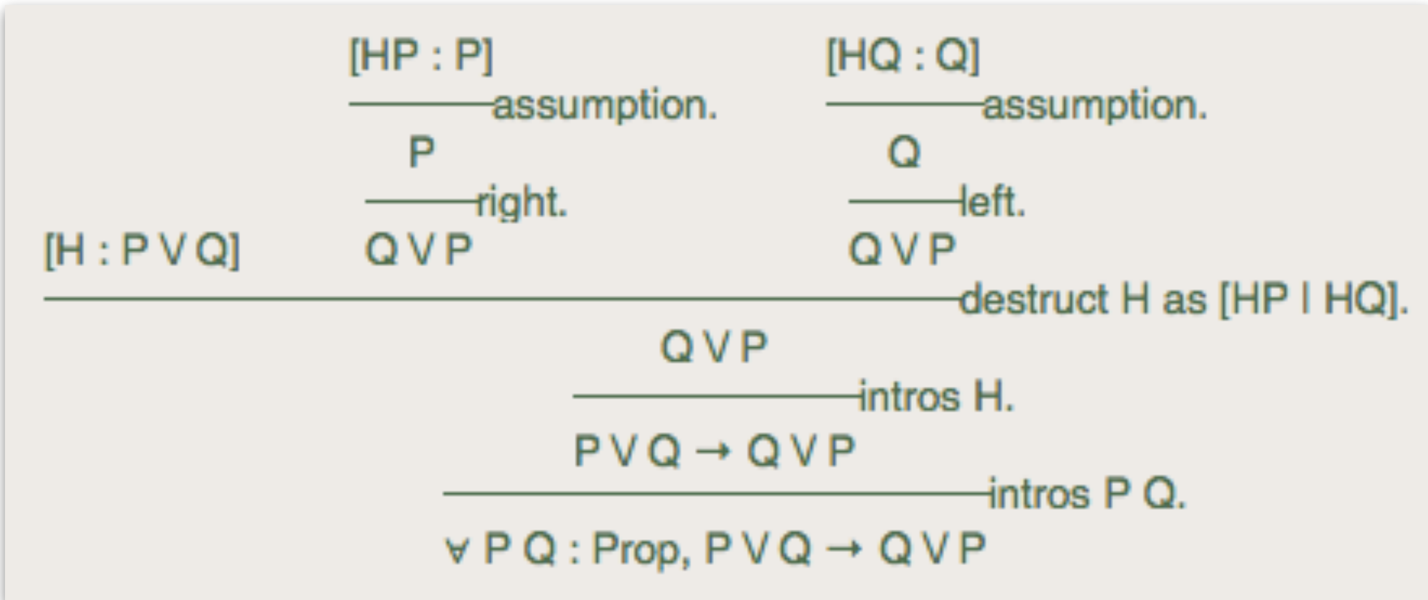
```
Theorem pq_qp: forall P Q: Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q.
  intros H.
  destruct H as [HP | HQ].
  right. assumption.
  left. assumption.
Qed.
```

$$\dfrac{\dfrac{\dfrac{[\mathrm{P} \lor \mathrm{Q}]^1 \qquad \dfrac{[\mathrm{P}]^2}{\mathrm{Q} \lor \mathrm{P}}\,(\lor\text{-intror}) \qquad \dfrac{[\mathrm{Q}]^2}{\mathrm{Q} \lor \mathrm{P}}\,(\lor\text{-introl})}{\mathrm{Q} \lor \mathrm{P}}\,(\lor\text{-elim})^2}{\mathrm{P} \lor \mathrm{Q} \;\to\; \mathrm{Q} \lor \mathrm{P}}\,(\to\text{-intro})^1}{\forall\ \mathrm{P}\ \mathrm{Q} : \mathrm{Prop}\ ,\ \mathrm{P} \lor \mathrm{Q} \;\to\; \mathrm{Q} \lor \mathrm{P}}\,(\forall\text{-intro})$$

Gentzen-style natural deduction proof

Visualizing a proof as a proof tree by Traf

```
Theorem pq_qp: forall P Q: Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q.
  intros H.
  destruct H as [HP | HQ].
  right. assumption.
  left. assumption.
Qed.
```



Traf's tree

Visualizing a proof as a proof tree by Traf

$$
\dfrac{\dfrac{\dfrac{[P \lor Q]^1 \qquad \dfrac{[P]^2}{Q \lor P}\,(\lor\text{-intror}) \qquad \dfrac{[Q]^2}{Q \lor P}\,(\lor\text{-introl})}{Q \lor P}\,(\lor\text{-elim})^2}{P \lor Q \;\to\; Q \lor P}\,(\to\text{-intro})^1}{\forall\ P\ Q : \mathrm{Prop}\ ,\ P \lor Q \;\to\; Q \lor P}\,(\forall\text{-intro})
$$

easy to read 😄

(if it is not too large)

burdensome 😣
to build

by using paper and pencil

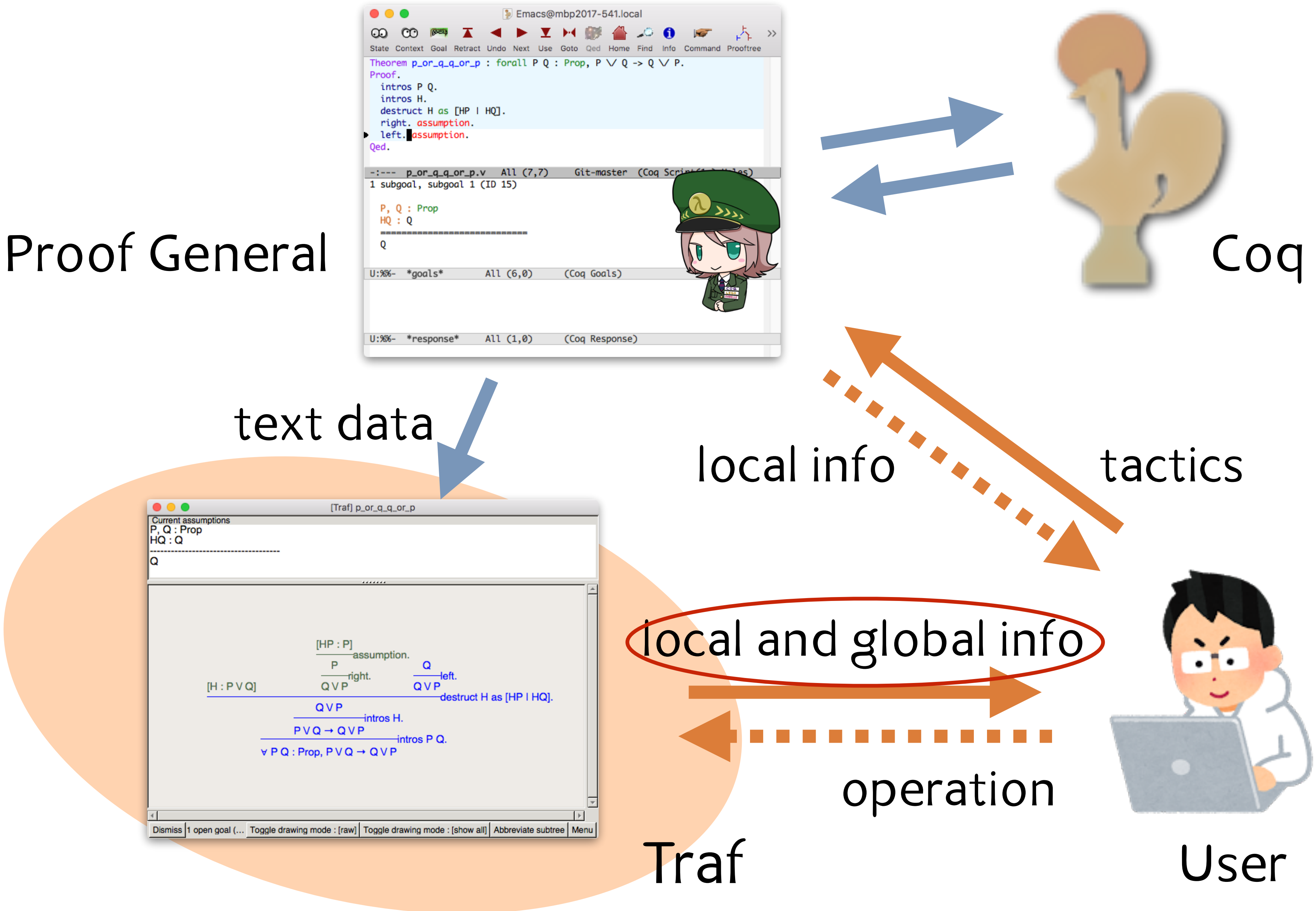
$$
\dfrac{\dfrac{\dfrac{[H : P \lor Q] \qquad \dfrac{\dfrac{[HP : P]}{P}\,\text{assumption.}}{Q \lor P}\,\text{right.} \qquad \dfrac{\dfrac{[HQ : Q]}{Q}\,\text{assumption.}}{Q \lor P}\,\text{left.}}{Q \lor P}\,\text{destruct H as [HP | HQ].}}{P \lor Q \to Q \lor P}\,\text{intros H.}}{\forall\ P\ Q : \mathrm{Prop},\ P \lor Q \to Q \lor P}\,\text{intros P Q.}
$$

easy to read 😄

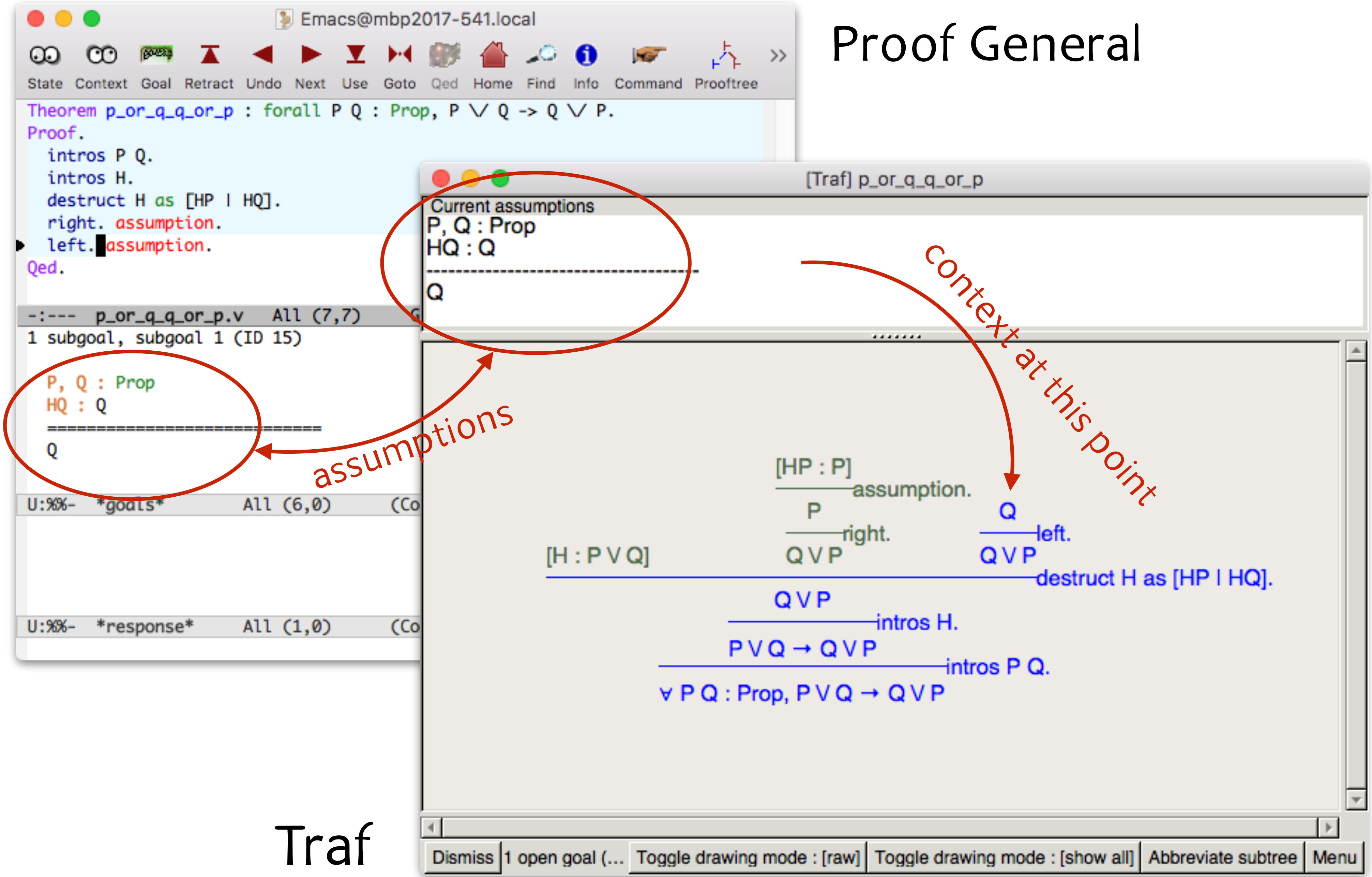
(if it is not too large)

no effort required
~~easy~~ to build 😄

Using Traf: General structure



Using Traf: General structure

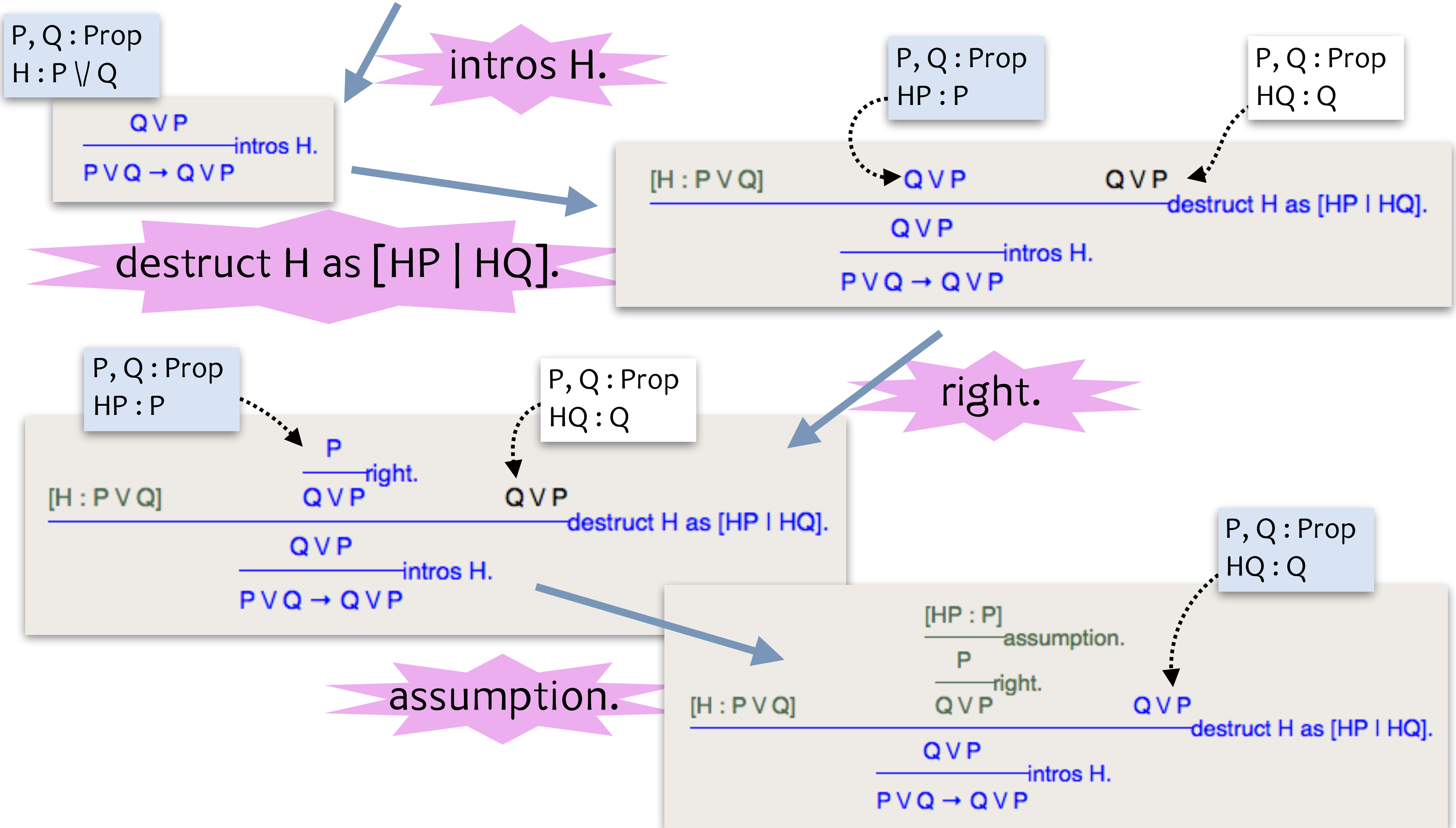


Proof General

Traf

Interactive construction of proof trees by Traf

Theorem p_or_q (P Q : Prop) : P ∨ Q → Q ∨ P.



Visualizing a proof as a proof tree by Traf

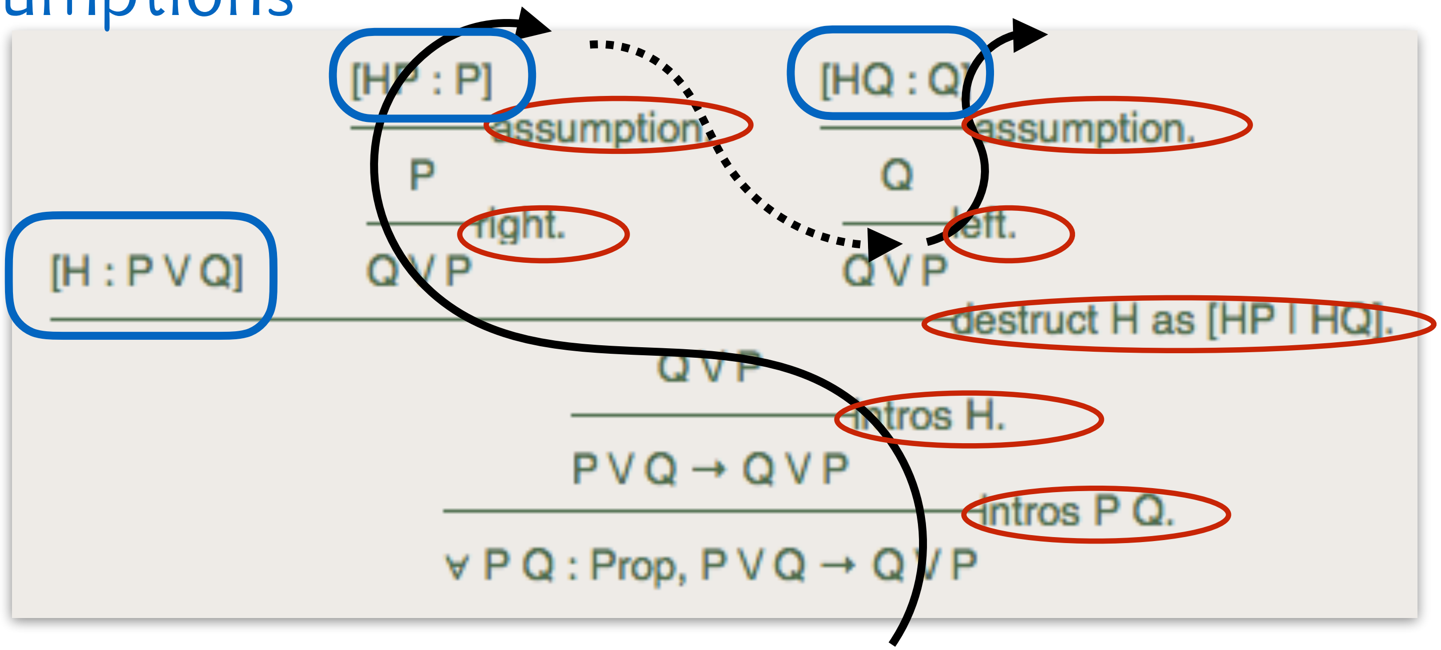
```
Theorem pq_qp: forall P Q: Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q.
  intros H.
  destruct H as [HP | HQ].
  right. assumption.
  left. assumption.
Qed.
```

grows as if it is performing
pre-order traversal
(like depth-first search)

all tactics are shown

Traf puts used assumptions with names

makes proof trees "readable"



Visualizing a proof as a proof tree by Traf

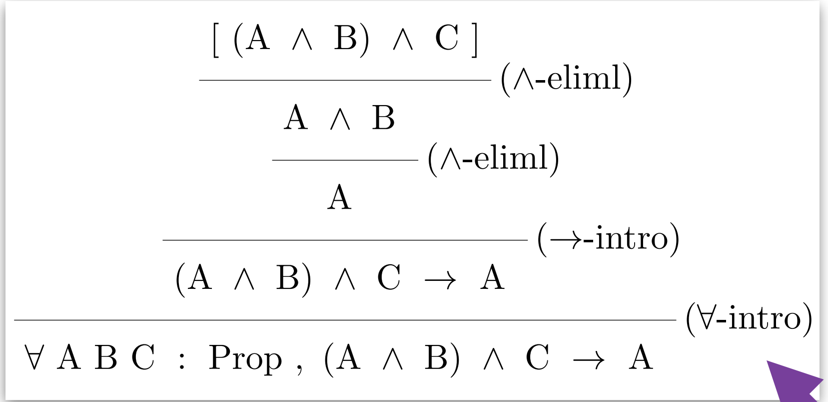
Another style of proof tree

```
                                      n : nat, IHn : n + 0 = n ⊢ S n = S n
  ───────reflexivity.                ──────────────────────────────────────rewrite -> IHn.
  ⊢ 0 = 0                            n : nat, IHn : n + 0 = n ⊢ S (n + 0) = S n
──────────simpl.                     ──────────────────────────────────────simpl.
⊢ 0 + 0 = 0                           n : nat, IHn : n + 0 = n ⊢ S n + 0 = S n
─────────────────────────────────────────────────────────────────────────induction n.
                        n : nat ⊢ n + 0 = n
                      ─────────────────────intros.
                      ⊢ ∀ n : nat, n + 0 = n
```

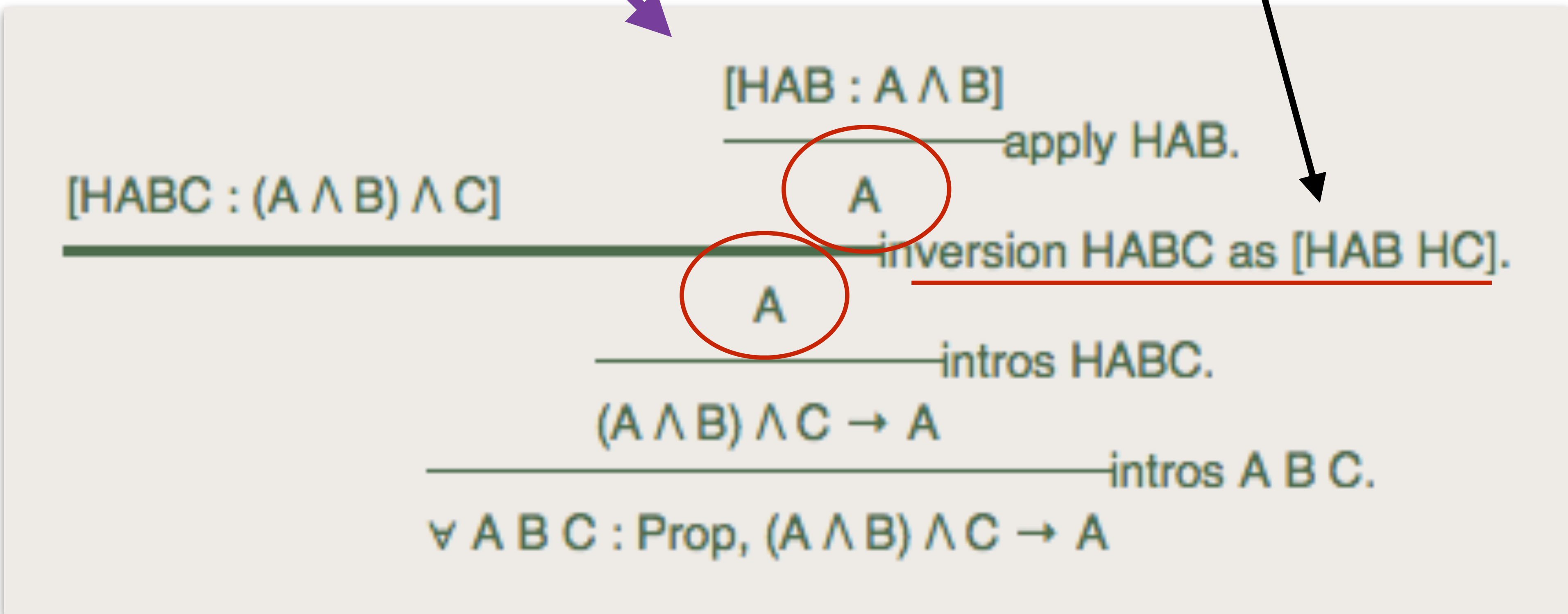
expressing everything could make the tree less readable

```
                                        [IHn : n + 0 = n]           S n = S n
                    ────reflexivity.    ─────────────────────────────────────rewrite → IHn.
                    0 = 0                                      S (n + 0) = S n
                  ──────simpl.                                 ───────────simpl.
[n : nat]         0 + 0 = 0                                     S n + 0 = S n
───────────────────────────────────────────────────────────────────────────induction n.
                                 n + 0 = n
                              ──────────────intros.
                              ∀ n : nat, n + 0 = n
```

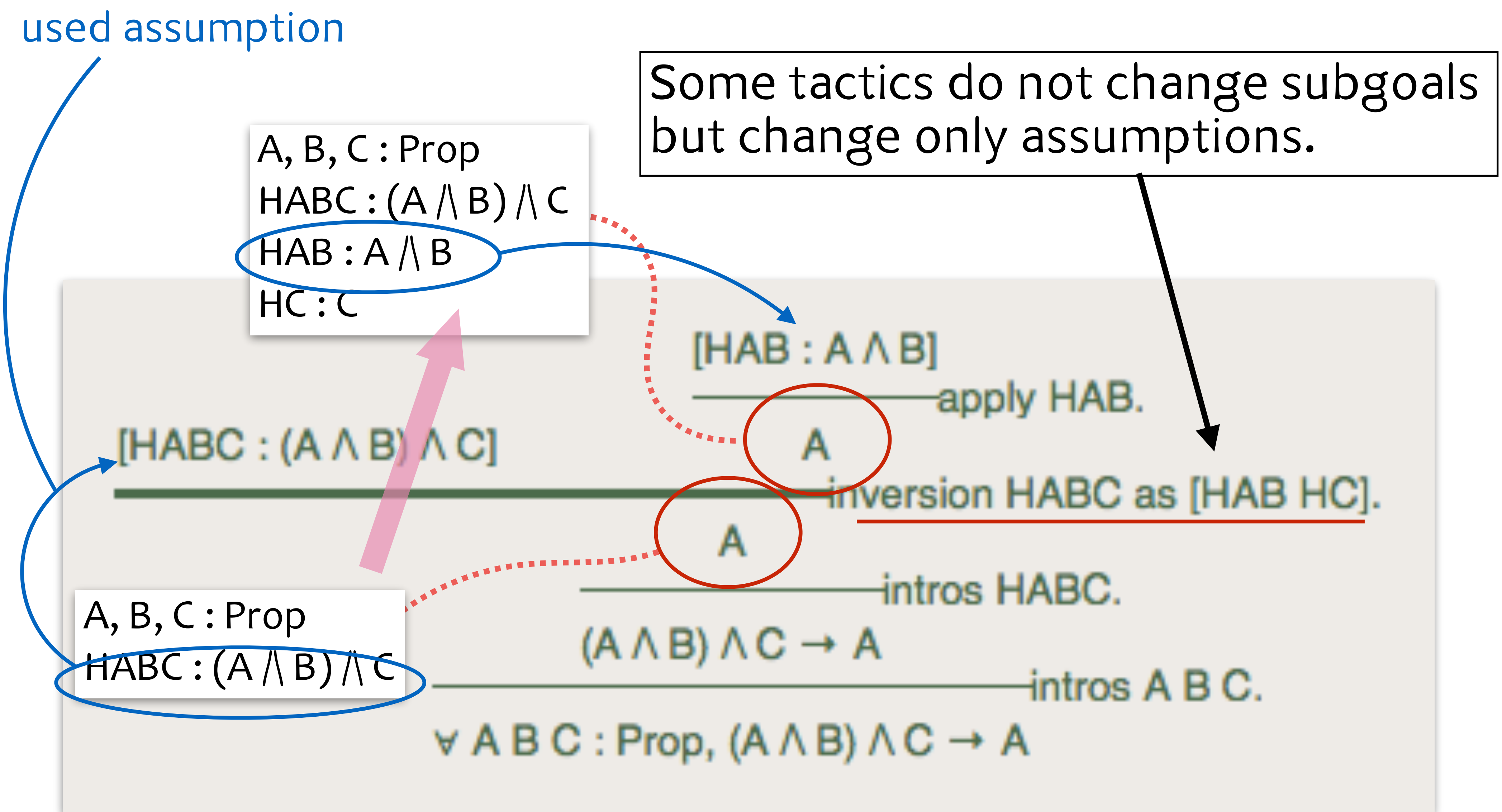
Visualizing a proof as a proof tree by Traf



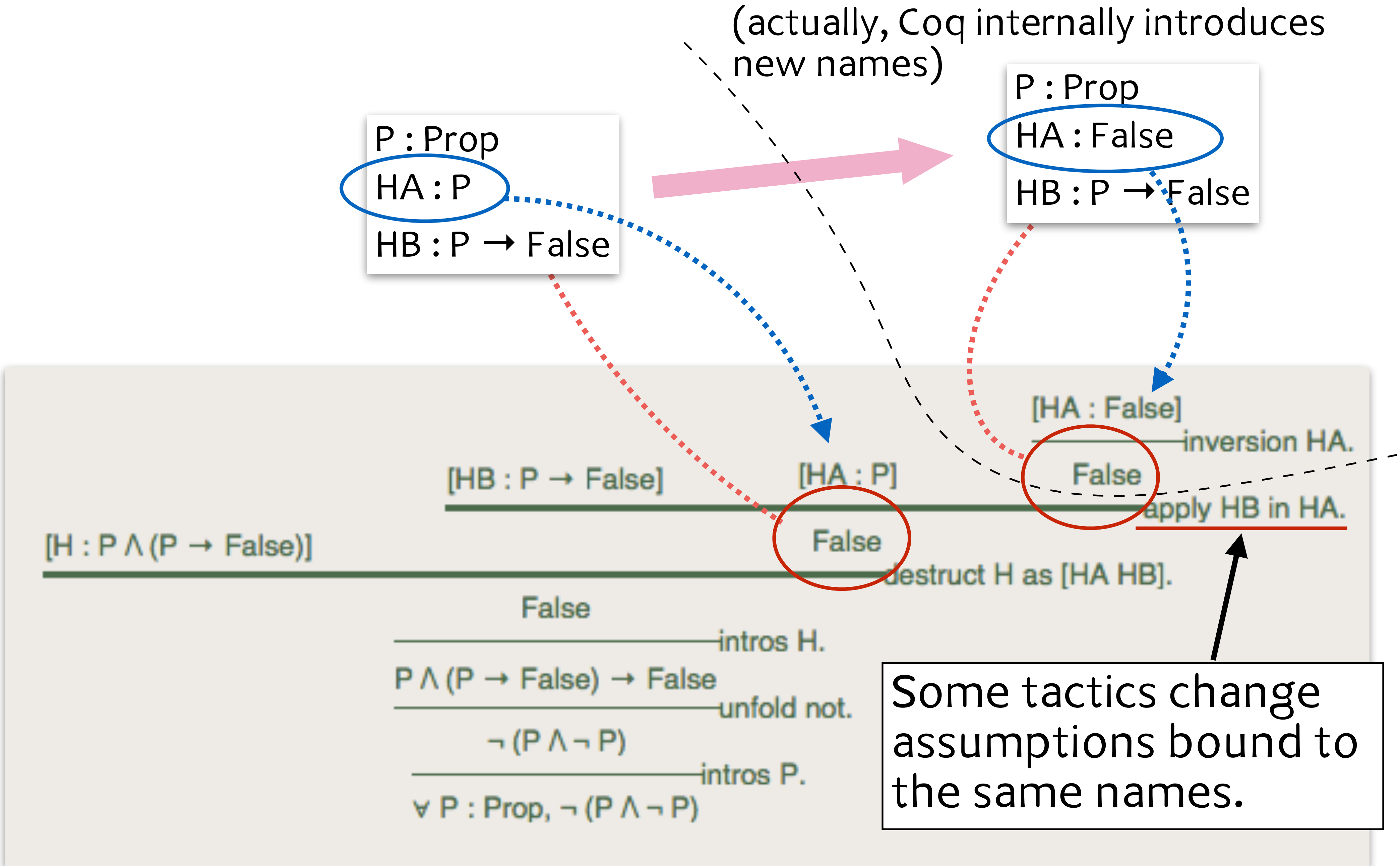
Some tactics do not change subgoals but change only assumptions.



Visualizing a proof as a proof tree by Traf



Visualizing a proof as a proof tree by Traf



Visualizing a proof as a proof tree by Traf

Use of tacticals and/or automation

$$\frac{}{P \vee Q \to Q \vee P}\text{intros H; destruct H; auto.}$$

sometimes they could make proof trees needless ...

$$\frac{}{\forall\, P\, Q\, R : \text{Prop}, (P \to Q) \to (Q \to R) \to P \to R}\text{auto.}$$

↕

```
(fun (P Q : Prop) (H : P \/ Q) =>
 match H with
 | or_introl H0 => or_intror H0
 | or_intror H0 => or_introl H0
 end)
```

In that case, corresponding proof object might be informative.
But, ...

```
(fun (P Q R : Prop) (H : P -> Q) (H0 : Q -> R) (H1 : P) => H0 (H H1))
```

Visualizing a proof as a proof tree by Traf

```
Theorem ceval_deterministic': forall c st st1 st2,
     c / st \\ st1  ->
     c / st \\ st2 ->
     st1 = st2.
Proof.
  intros c st st1 st2 E1 E2;
  generalize dependent st2;
  induction E1;
           intros st2 E2; inv E2; repeat find eqn; try find rwinv; auto.
Qed.
```

in Auto.v, Software Foundations

One-line proof script

```
(fun (c : com) (st st1 st2 : state) (E1 : c / st \\ st1) (E2 : c / st \\ st2)
 =>
 ceval_ind
  (fun (st0 : state) (c0 : com) (st3 : state) =>
   forall st4 : state, c0 / st0 \\ st4 -> st3 = st4)
  (fun (st0 st3 : state) (E3 : SKIP / st0 \\ st3) =>
   let H : st0 = st0 -> SKIP = SKIP -> st3 = st3 -> st0 = st3 :=
    match
     E3 in (c0 / s \\ s0)
     return (s = st0 -> c0 = SKIP -> s0 = st3 -> st0 = st3)
    with
    | E_Skip st4 =>
      fun (H : st4 = st0) (H0 : SKIP = SKIP) (H1 : st4 = st3) =>
  ...
```

Corresponding proof object (› 2500 lines)

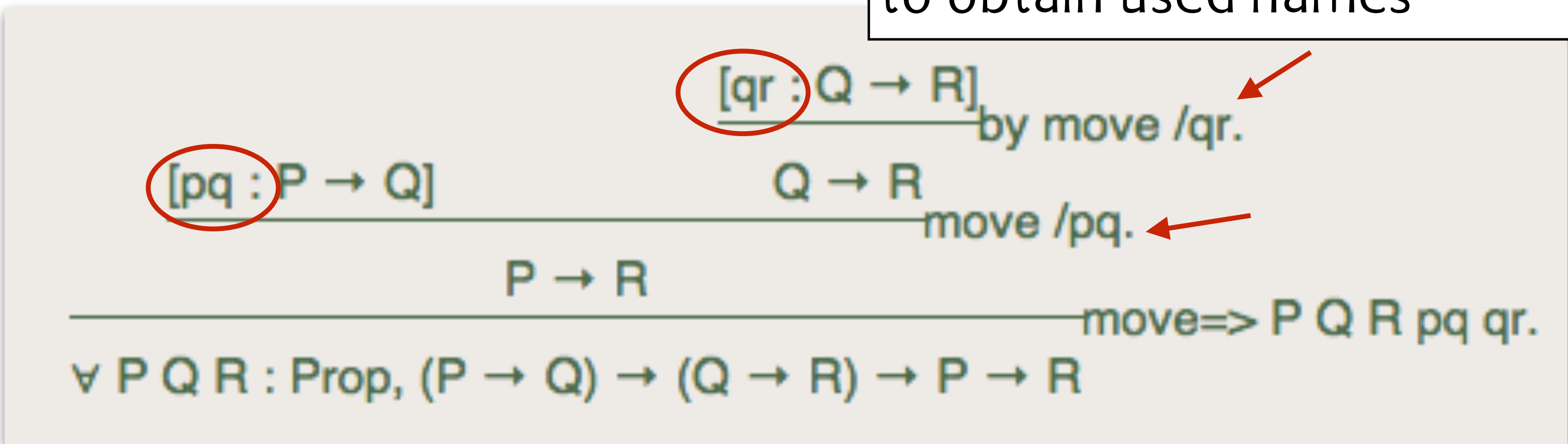
It depends.

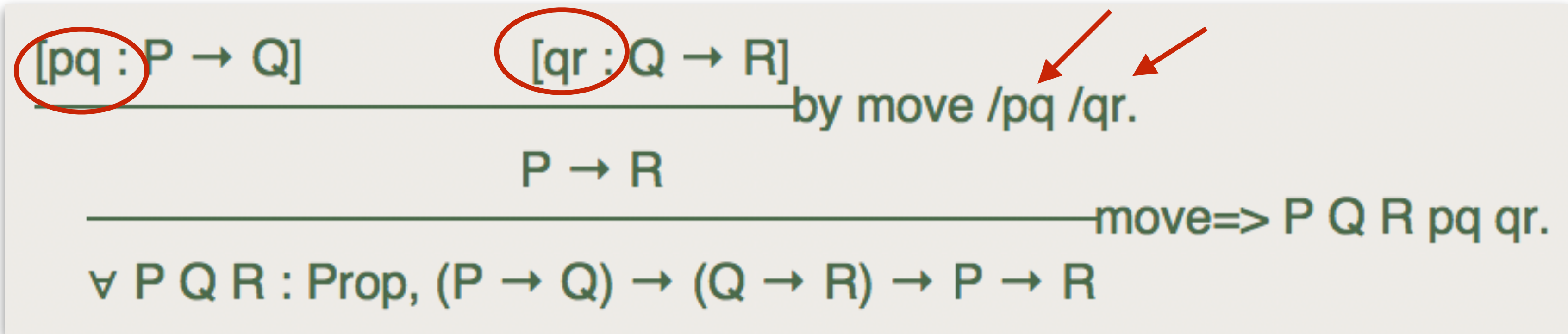
(Traf currently does not look at proof objects)

Visualizing a proof as a proof tree by Traf

Use of SSRefrect

Traf parses SSReflect's commands to obtain used names



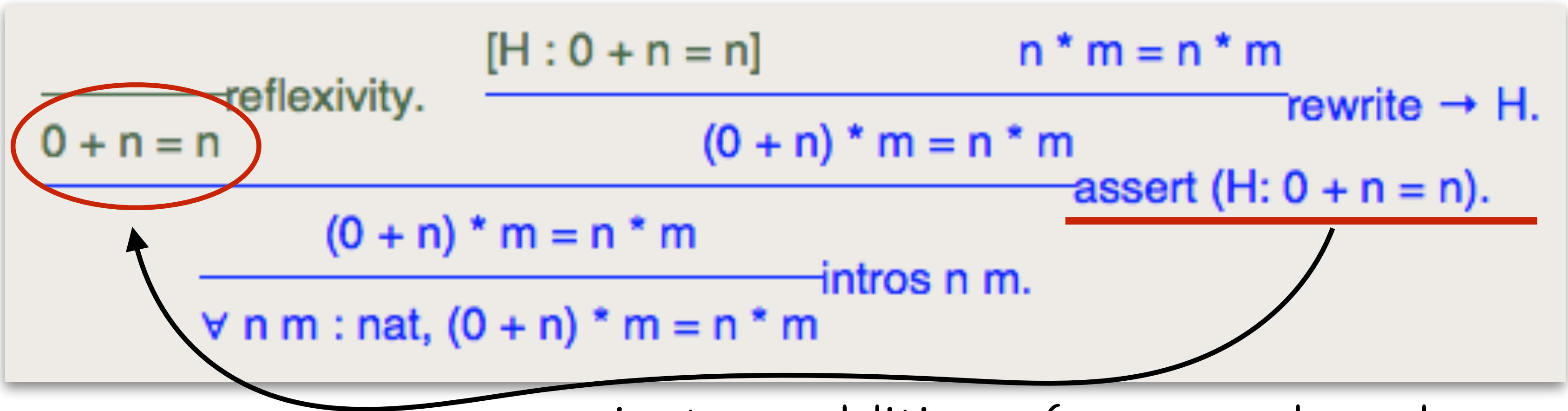


Traf recognizes used assumptions by analyzing commands syntactically.

- Adapting Traf for another tactic library only requires an additional parser for commands.

Visualizing a proof as a proof tree by Traf

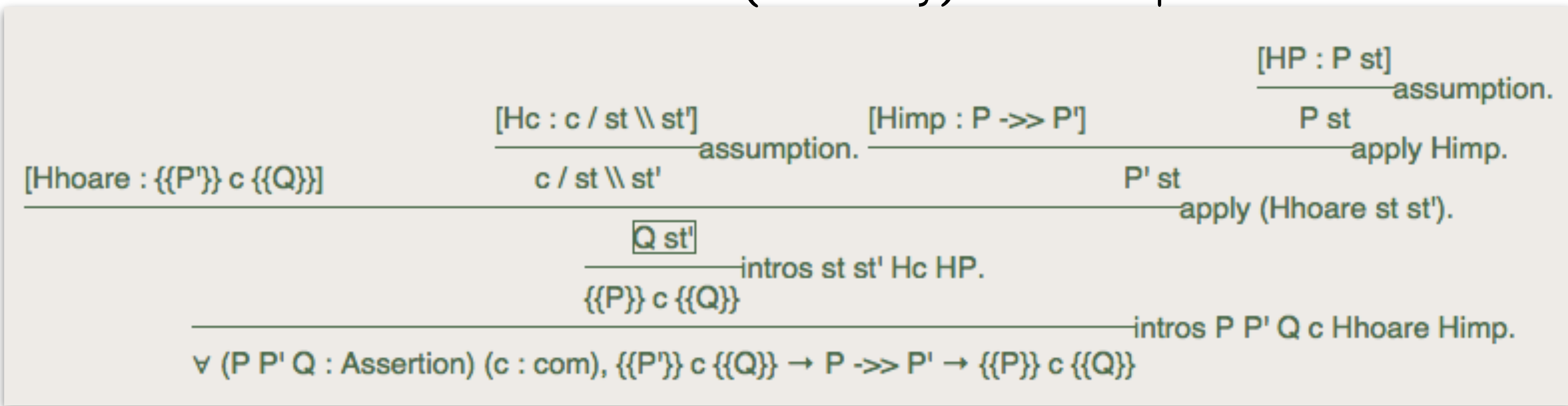
Use of `assert`



just an addition of a new subgoal

Use of special notations

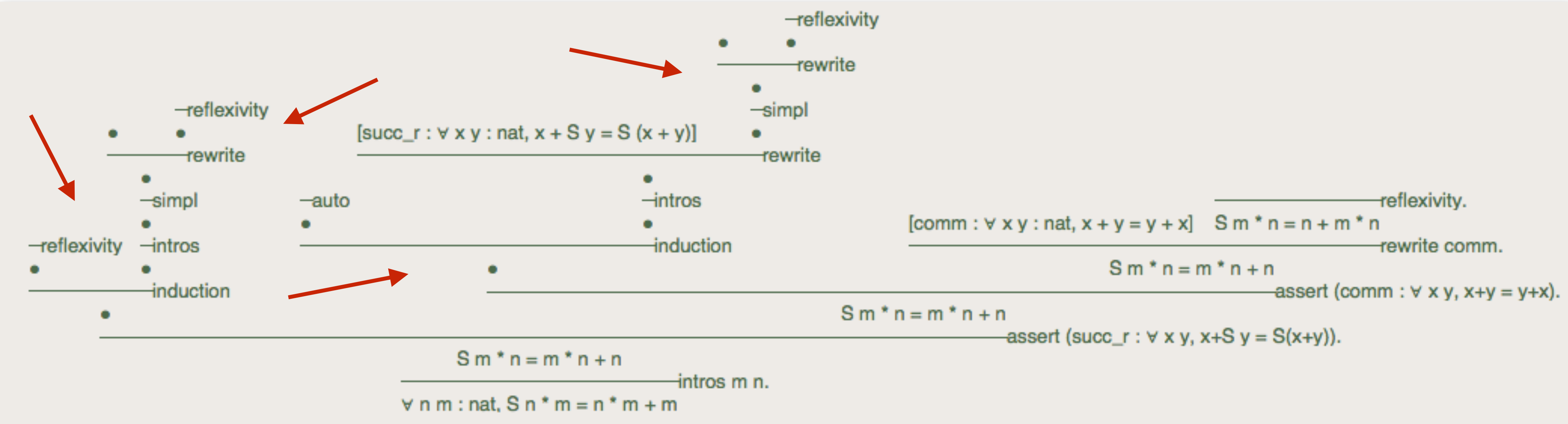
Traf (currently) does not parse each formula



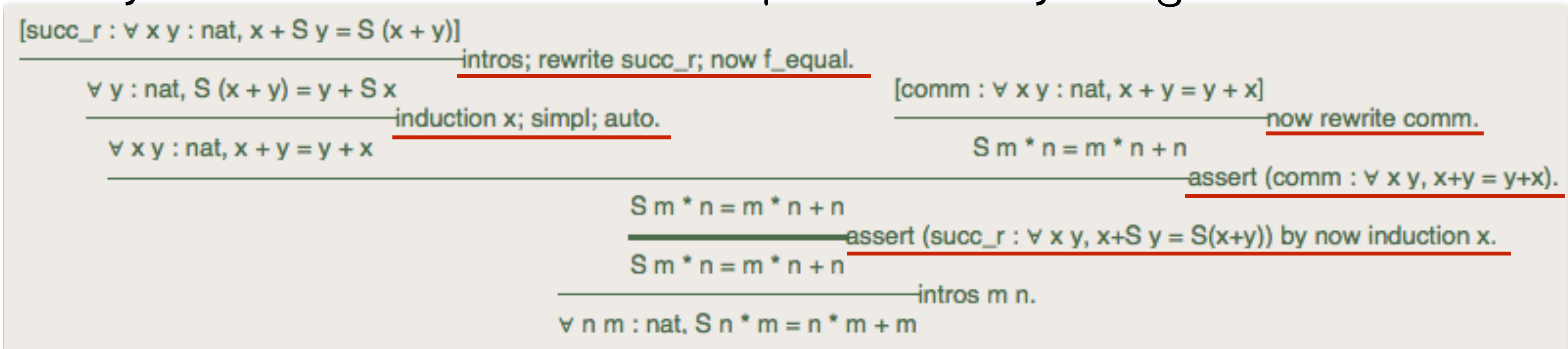
Visualizing a proof as a proof tree by Traf

What if the tree becomes large ?

- You can shrink nodes that are not very interesting.

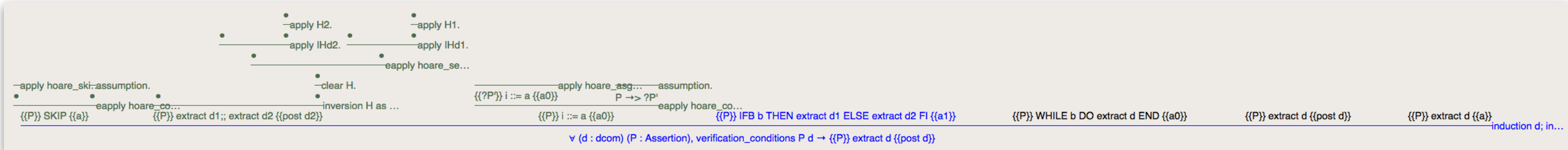


- Or you would like to shorten the proof itself by using tacticals.



Visualizing a proof as a proof tree by Traf

What if the tree becomes very large ?



(scroll bars are available)

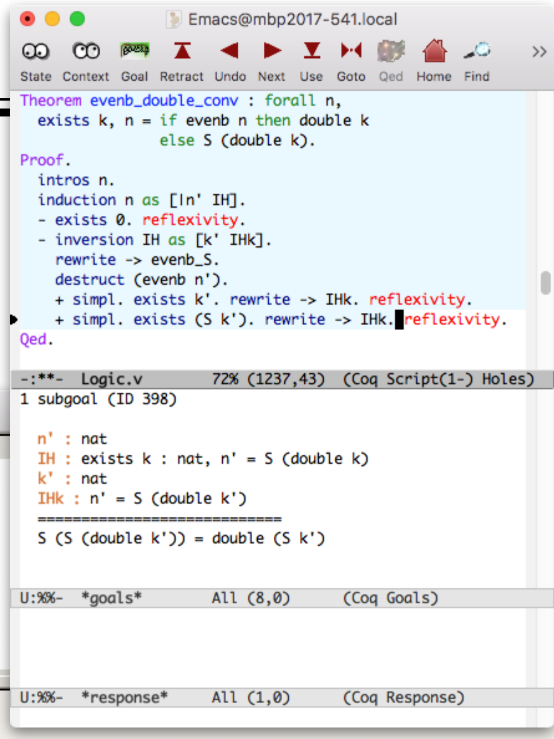
```
∀ (d : dcom) (P : Assertion), verification_conditions P d → {{P}} extract d {{post d}}.
|  induction d; intros P H; simple in *.
|-- {{P}} SKIP {{a}}
|  |  eapply hoare_consequence_pre.
|  |-- {{?P'}} SKIP {{a}}
|  |   | apply hoare_skip.
|  |   \----
|  \-- P ->> ?P'
|      | assumption.
|      \----
|-- {{P}} extract d1 ;; extract d2 {{post d2}}
|  |  inversion H as [H1 H2].
|  |-- [H : verification_conditions P d1 /\ verification_conditions (post d1) d2]
|  \-- {{P}} extract d1 ;; extract d2 {{post d2}}
|      | clear H.
|      \-- {{P}} extract d1 ;; extract d2 {{post d2}}
|          | eapply hoare_seq.
|          |-- {{?Q}} extract d2 {{post d2}}
|          |   | apply IHd2.
|          |   |---- [[IHd2 : ∀ P : Assertion, verification_conditions P d2 → {{P}} extract d2 {{post d2}}]
|          |   \-- verification_conditions ?Q d2
|          |       | apply H2.
|          |       \---- [H2 : verification_conditions (post d1) d2]
|          \-- {{P}} extract d1 {{?Q}}
|              | apply IHd1.
|              |---- [[IHd1 : ∀ P : Assertion‹ verification_conditions P d1 → {{P}} extract d1 {{post d1}}]
|              \-- verification_conditions P d1
```

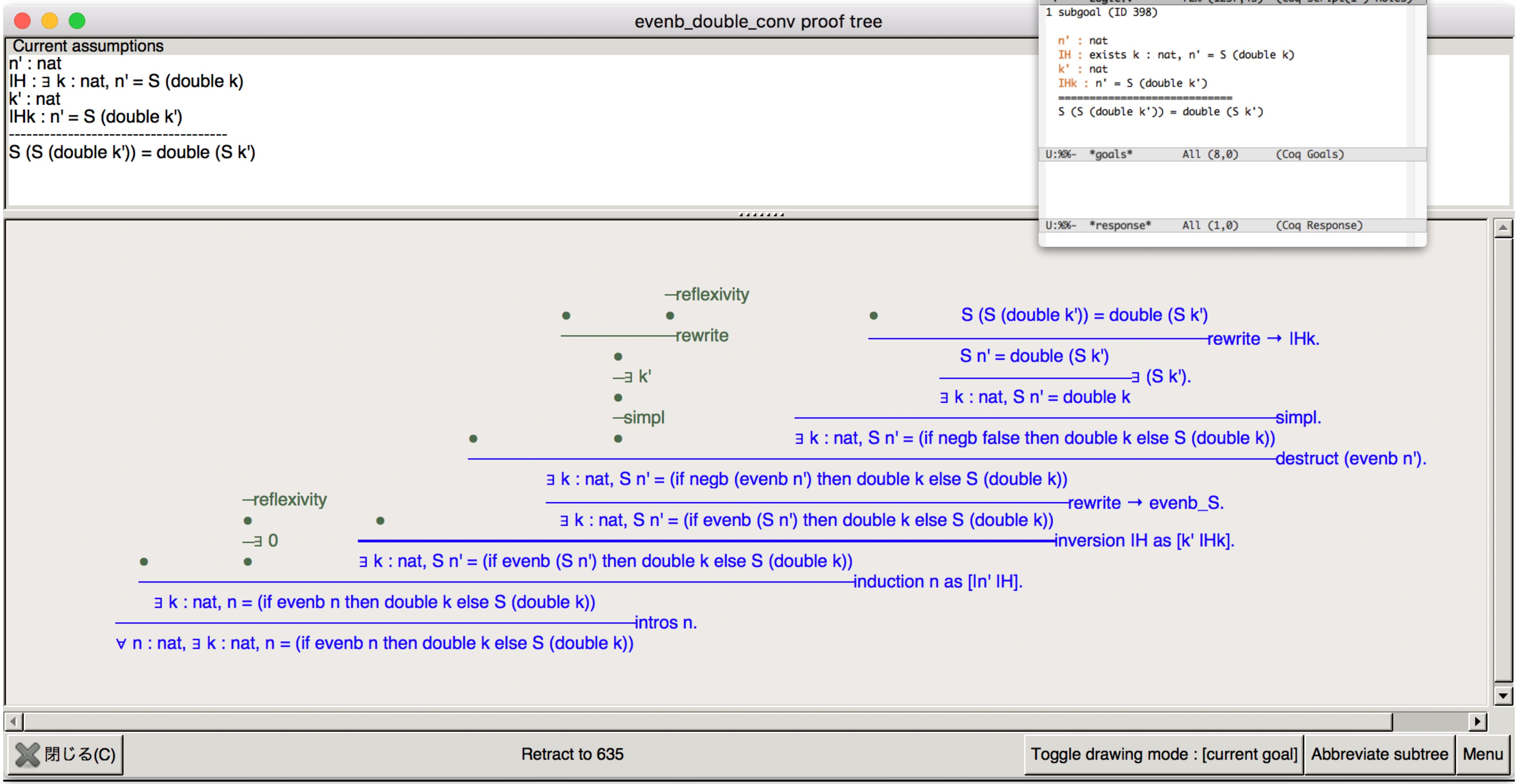
Turning the tree clockwise by 90 degrees would be preferable.
(not supported yet)

Traf as a Proving Situation Monitor

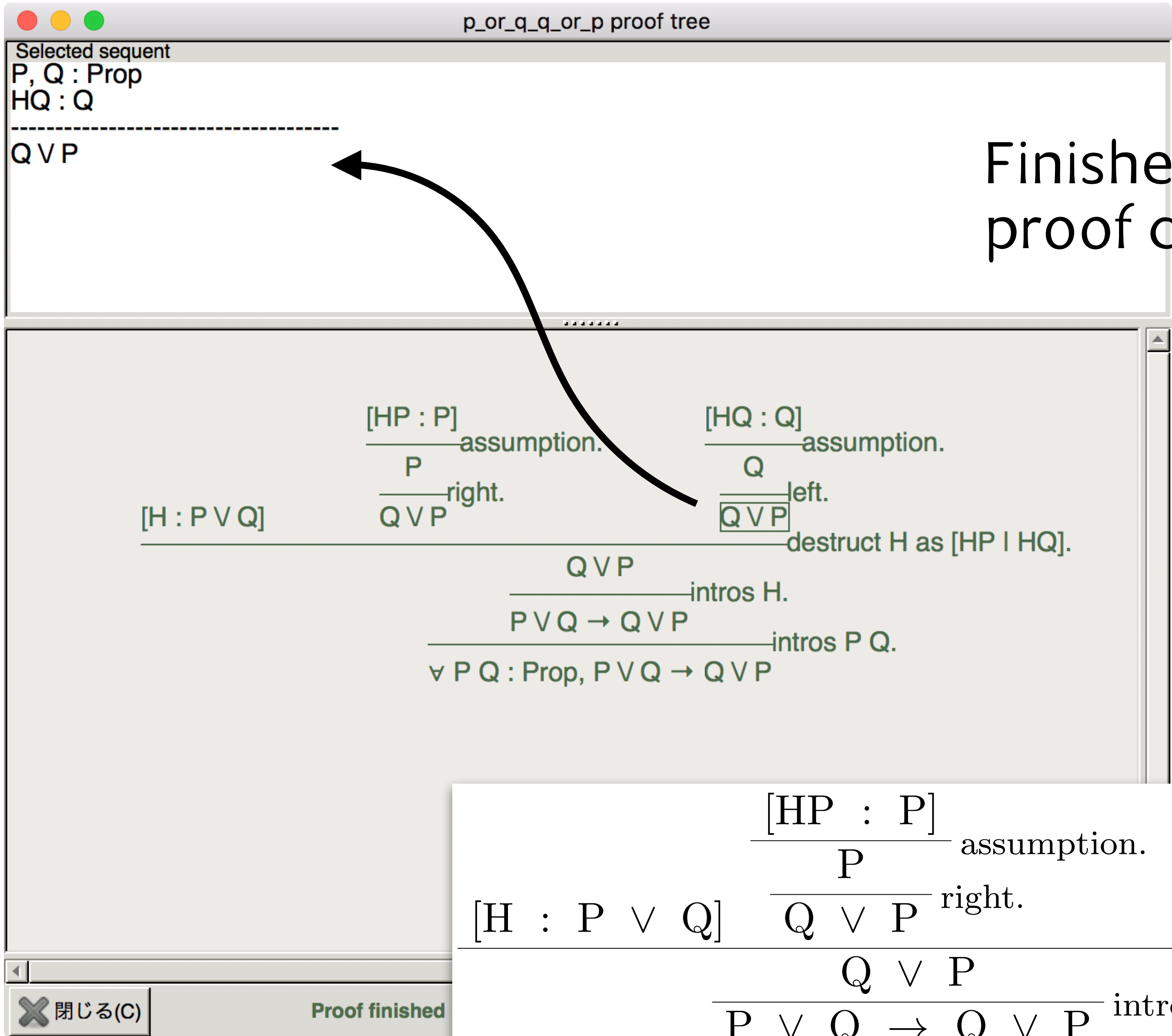
"Current Goal" mode

- automatically shrinks irrelevant nodes
- keeps the tree's size modest



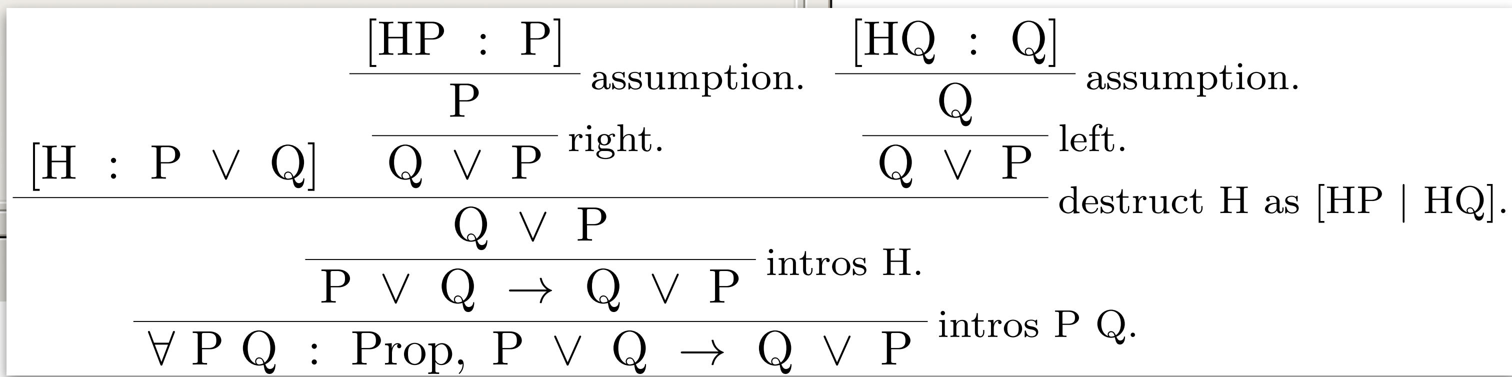


Traf as a Proof Script Browser



Finished (or unfinished) proof can be investigated

- tool tip
- LaTeX script generation

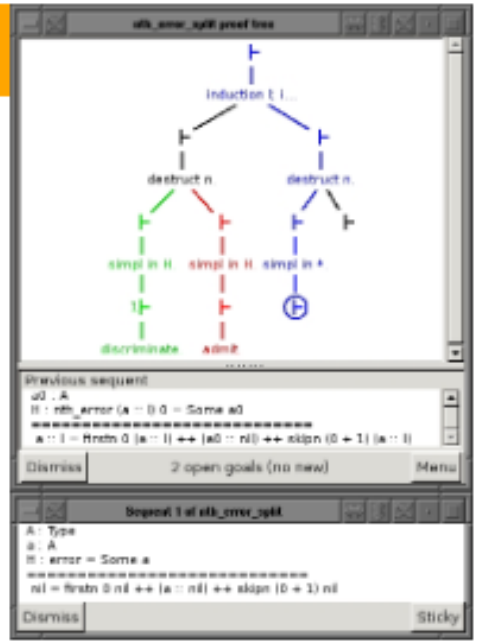


Traf is based on Prooftree

askra.de/software/prooftree/

Proof tree visualization for Proof General

Proof tree visualization for Proof General



Prooftree is a program for proof-tree visualization during interactive proof development in a theorem prover. It is currently being developed for Coq and Proof General. Prooftree helps against getting lost between different subgoals in interactive proof development. It clearly shows where the current subgoal comes from and thus helps in developing the right plan for solving it.

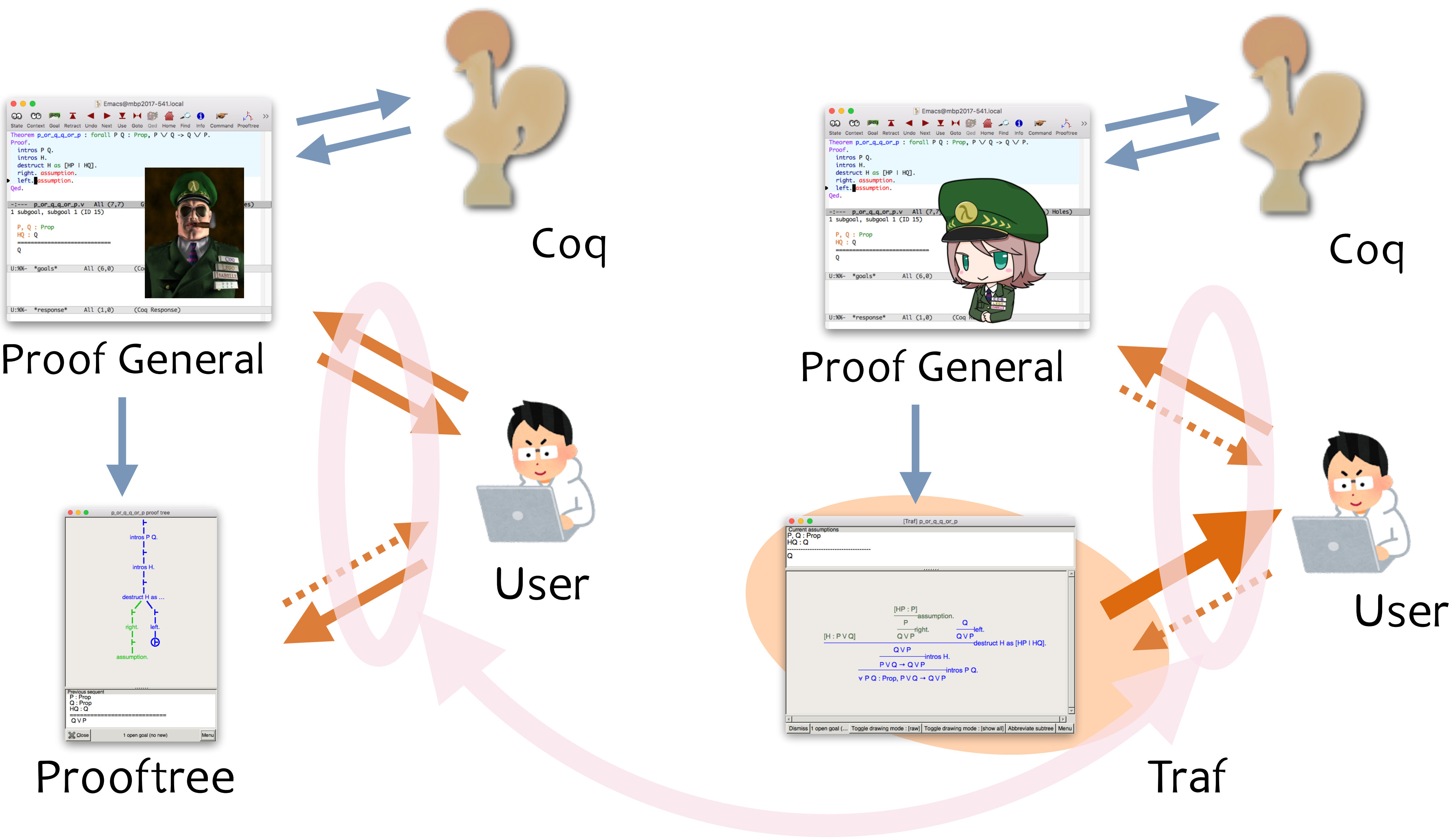
Prooftree uses different colors for the already proven subgoals, the current branch in the proof and the still open subgoals. Sequent texts are not displayed in the proof tree itself, but they are shown as a tool-tip when the mouse rests over a sequent symbol. Long proof commands are abbreviated in the tree display, but show up in full length as tool-tip. Both, sequents and proof commands, can be shown in the display below the tree (on single click) or in a separate window (on double or shift-click).

Prooftree can mark the proof command that introduced a certain existential variable and thus help to locate the problem when Coq says No more subgoals but non-instantiated existential variables.

Currently, prooftree does only work for Coq proofs. Adding support for a different proof assistant should not be too hard, see the Proof General Adapting manual. Please drop me a line, if you would like to help with another proof assistant.

Hendrik Tews: Prooftree homepage https://askra.de/software/prooftree/

Comparison: Prooftree and Traf



Both are implemented in OCaml with LablGtk library.

Conclusion

Traf: a Graphical Proof Tree Viewer Cooperating with Coq through Proof General

Available at https://github.com/hide-kawabata/traf

Our approach:

graphically

provide support for writing and reading proof scripts

"interactive monitor"

"proof script browser"

for Proof General users

Future work

- support for large proof, long commands and/or tacticals, automation, ...
- support for referencing external lemmas, ...
- (evaluation based on user study)