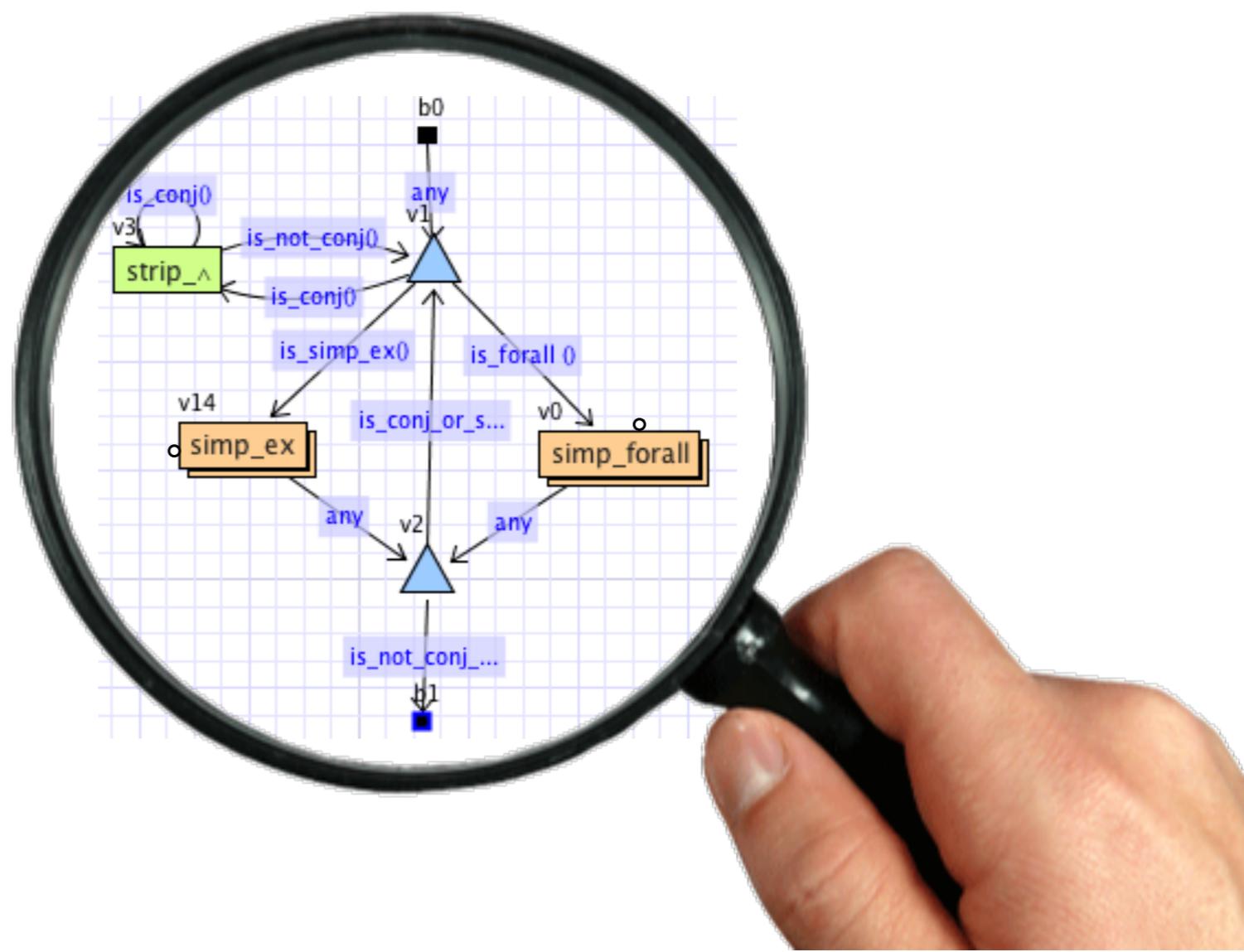


# Developing & Debugging Proof Strategies

by **TINKERING**

Yuhui Lin,  
Pierre Le Bras,  
Gudmund Grov

April 2016, TACAS  
Eindhoven



# Stack-based strategies

LCF-style provers operate on open goals using **tactics**:

**t: goal -> [goal]**

Proof strategies are built from tactics using **tactical** combinators

# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$

mytac(g) :=

|

# Stack-based strategies

**tac** mytac := t<sub>1</sub> THEN t<sub>2</sub> THEN t<sub>2</sub> THEN t<sub>3</sub>



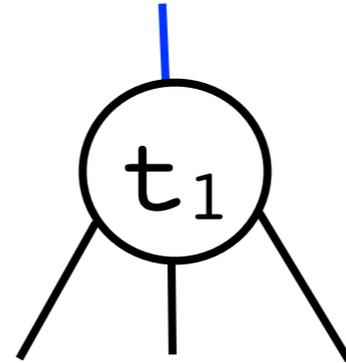
mytac(g) :=



# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$

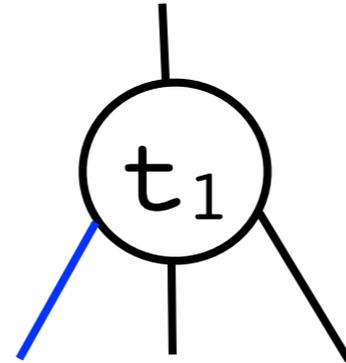
mytac(g) :=



# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$

mytac(g) :=

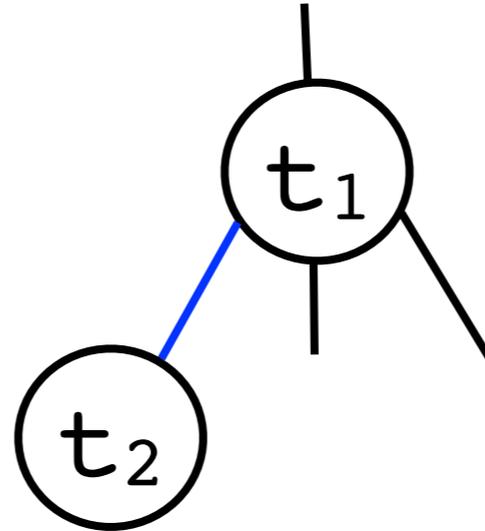


# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$



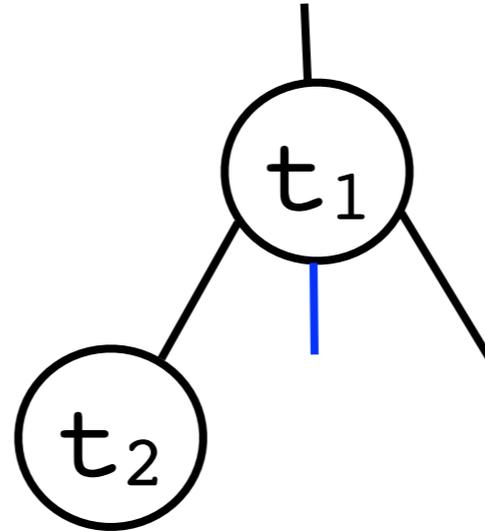
mytac(g) :=



# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$

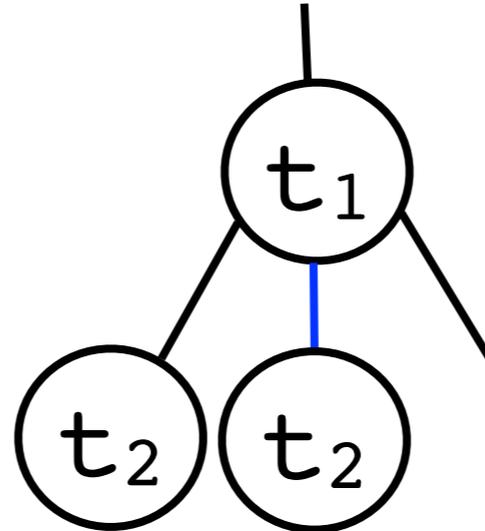
mytac(g) :=



# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$

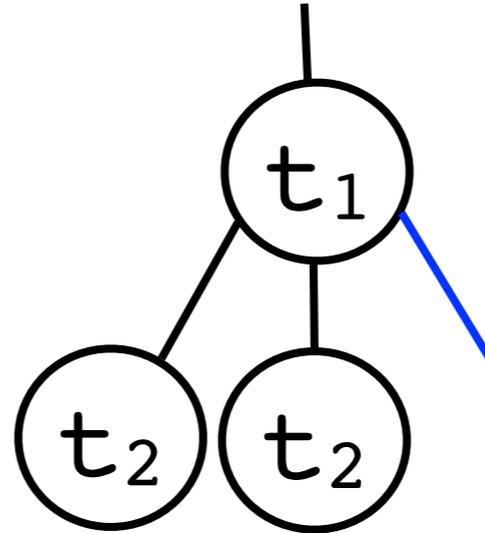
mytac(g) :=



# Stack-based strategies

**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$   
↑

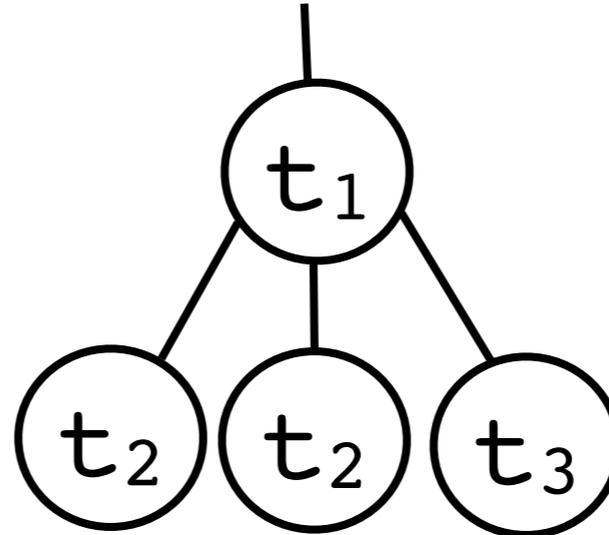
mytac(g) :=



# Stack-based strategies

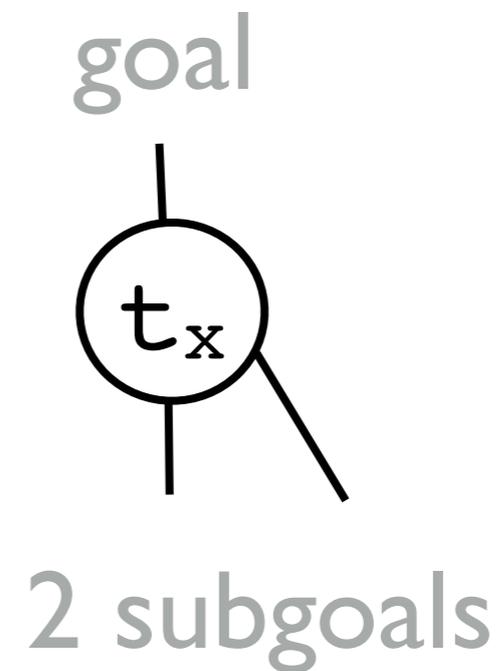
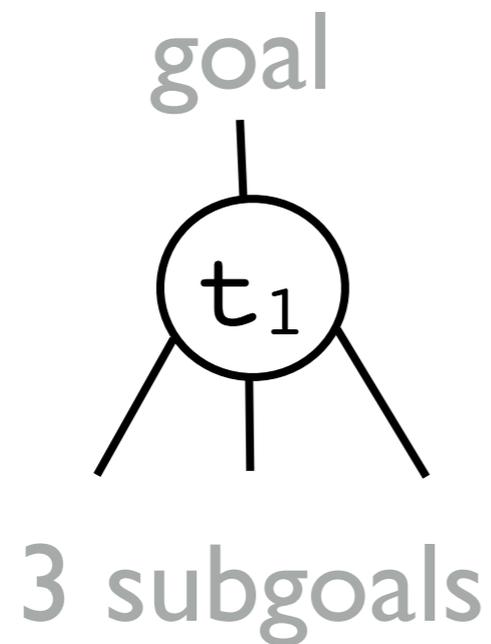
**tac** mytac :=  $t_1$  THEN  $t_2$  THEN  $t_2$  THEN  $t_3$

mytac(g) :=



# But sometimes it goes wrong....

Suppose we replace  $t_1$  with the “improved” tactic  $t_x$



# Debugging

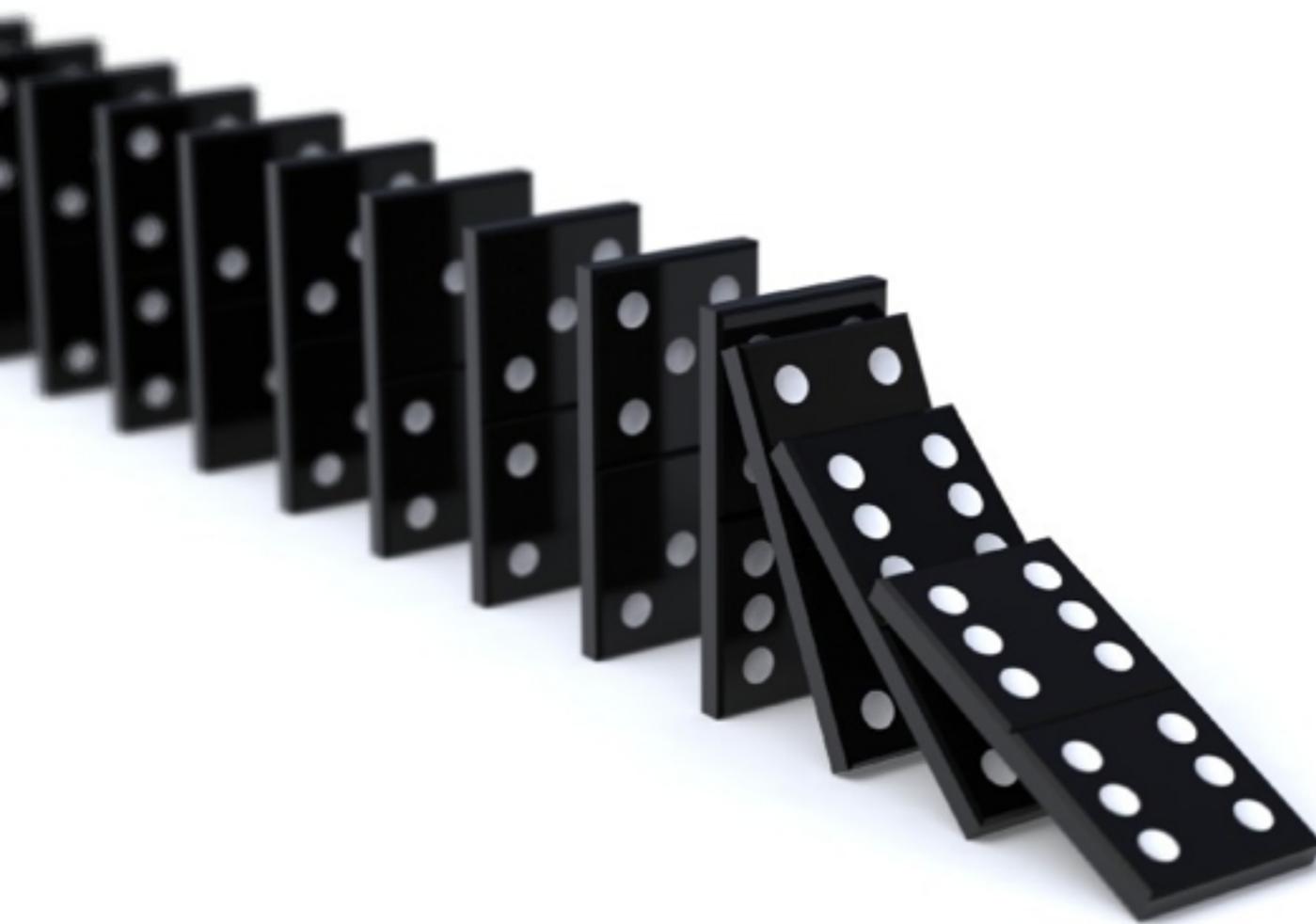
where did it go wrong?

actual error

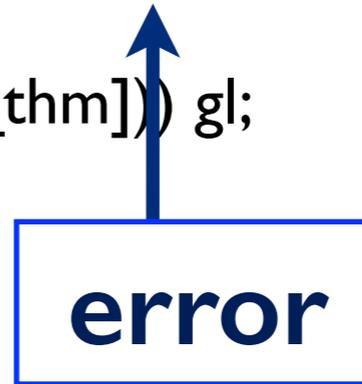
here or here

`tac mytac := tx THEN t2 THEN t2 THEN t3`

error



```
(in thm rewrite_tac thms end
  handle (Fail _) => thm)) o rev) THEN
(TRY_T (rewrite_tac thms)) THEN
REPEAT strip_tac THEN
TRY_T all_var_elim_asm_tac THEN_TRY
(z_quantifiers_elim_tac THEN
(fn gl => let val ciz = set_check_is_z false;
val res = (EXTEND_PC_T1 "mmp1" all_asm_fc_tac[]) THEN
  (basic_res_tac2 3 [eq_refl_thm]
  ORELSE_T basic_res_tac3 3 [eq_refl_thm])) gl;
val _ = set_check_is_z ciz; in res end
));
```



```

TRY_T all_var_elim_asm_tac THEN
DROP_ASMS_T (MAP EVERY (strip_asm_tac o
(fn thm => rewrite_rule thms thm
  handle (Fail _) => thm)) o rev) THEN
(TRY_T (rewrite_tac thms)) THEN
REPEAT strip_tac THEN
TRY_T all_var_elim_asm_tac THEN_TRY
(z_quantifiers_elim_tac THEN
(fn gl => let val ciz = set_check_is_z false;
  (basic_res_tac2 3 [eq_refl_thm]
  ORELSE_T basic_res_tac3 3 [eq_refl_thm])) gl;
  val _ = set_check_is_z ciz; in res end
(fn thm => rewrite_rule thms thm
  handle (Fail _) => thm)) o rev) THEN
(TRY_T (rewrite_tac thms)) THEN
REPEAT strip_tac THEN
TRY_T all_var_elim_asm_tac THEN_TRY
(z_quantifiers_elim_tac THEN
(fn gl => let val ciz = set_check_is_z false;
  (basic_res_tac2 3 [eq_refl_thm]
  ORELSE_T basic_res_tac3 3 [eq_refl_thm])) gl;
  val _ = set_check_is_z ciz; in res end
(fn thm => rewrite_rule thms thm
  handle (Fail _) => thm)) o rev) THEN
(TRY_T (rewrite_tac thms)) THEN
REPEAT strip_tac THEN

```

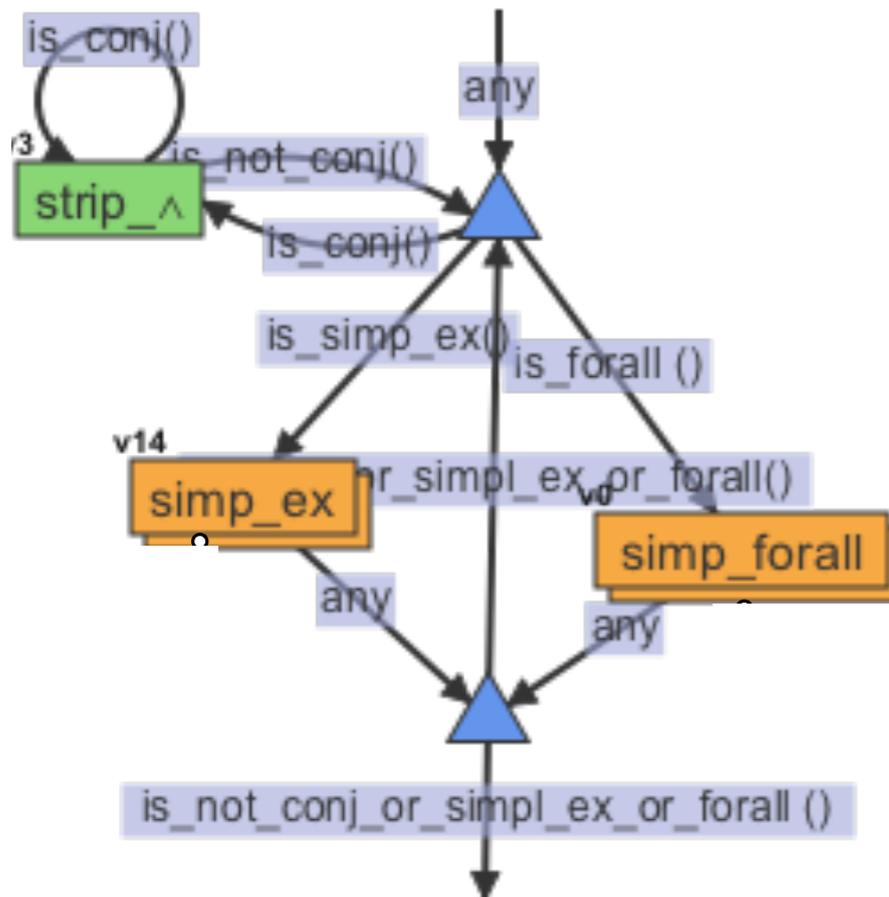
```

run z_basic_prove_tac (thms: Thm list).17/18/18 (
  TRY_T all_var_elim_asm_tac THEN
  DROP_ASMS_T (MAP EVERY (strip_asm_tac o
    (fn thm => rewrite_rule thms thm
      handle (Fail _) =>
        (TRY_T (rewrite_tac thms) THEN
          REPEAT strip_tac THEN
            TRY_T all_var_elim_asm_tac THEN
              (z_quantifiers_elim_tac THEN
                (fn gl => let val ciz =
                  (basic_res_tac2
                    ORELSE_T basic_res_tac3
                      val _ = set_check_is_z ciz; in res end
                (fn thm => rewrite_rule thms thm
                  handle (Fail _) => thm)) o rev) THEN
                (TRY_T (rewrite_tac thms) THEN
                  REPEAT strip_tac THEN
                    TRY_T all_var_elim_asm_tac THEN TRY
                    (z_quantifiers_elim_tac THEN
                      (fn gl => let val ciz = set_check_is_z false;
                        (basic_res_tac2 3 [eq_refl_thm]
                          ORELSE_T basic_res_tac3 3 [eq_refl_thm])) gl;
                        val _ = set_check_is_z ciz; in res end
                      (fn thm => rewrite_rule thms thm
                        handle (Fail _) => thm)) o rev) THEN
                        (TRY_T (rewrite_tac thms) THEN
                          REPEAT strip_tac THEN

```



# Tinker



- proof strategies as hierarchical graphs
- tactic composition
  - by connecting nodes with edges
- edges with goal types
- tactic application
  - by consuming and producing goals through nodes

# Tinker Tool

## Nodes:

tactic node

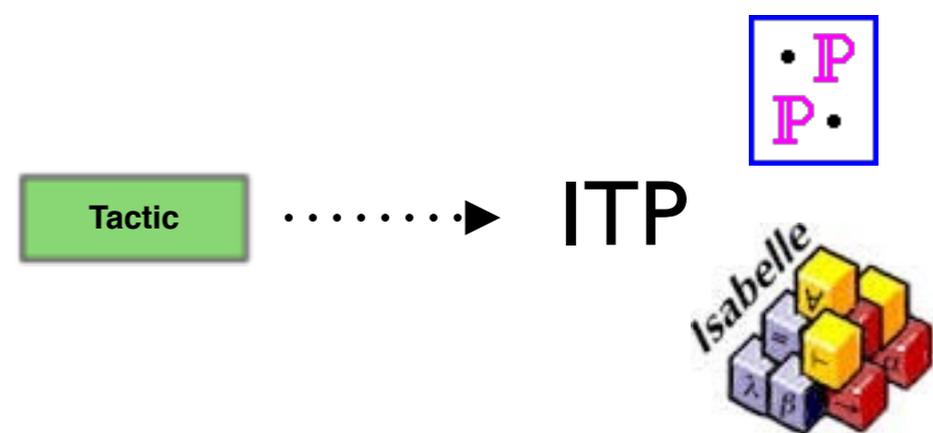
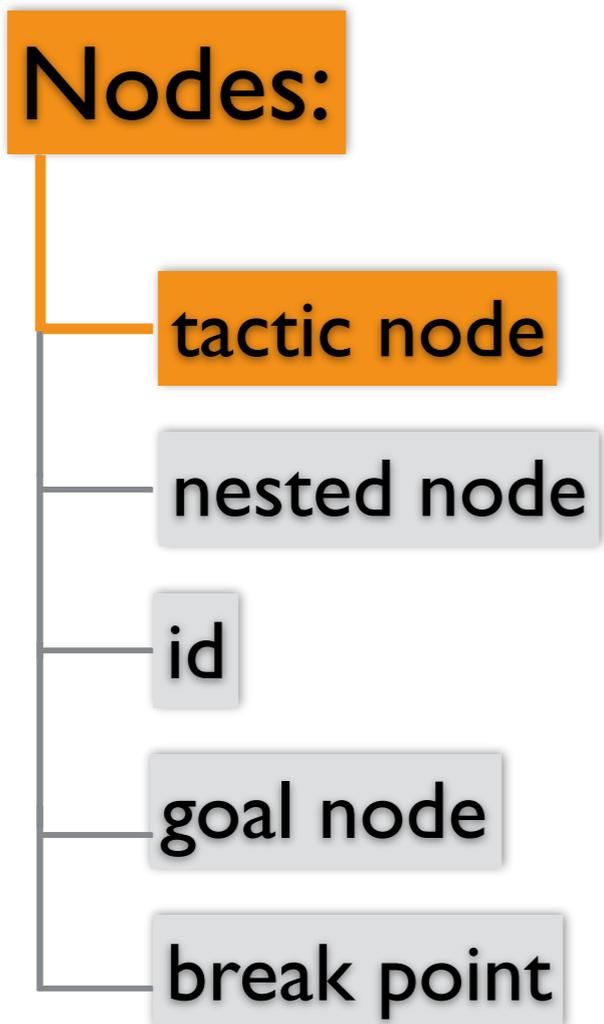
nested node

id

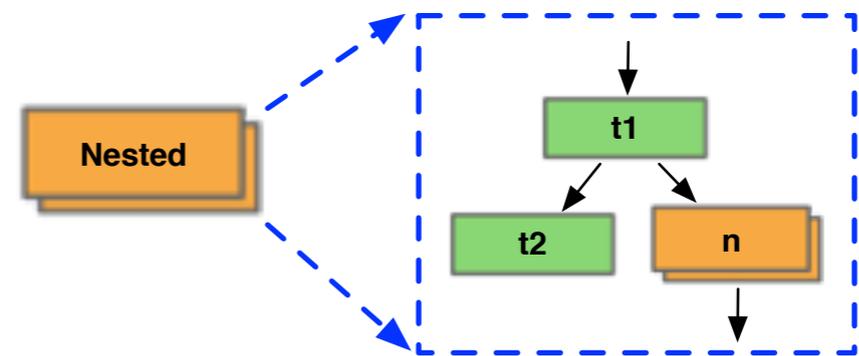
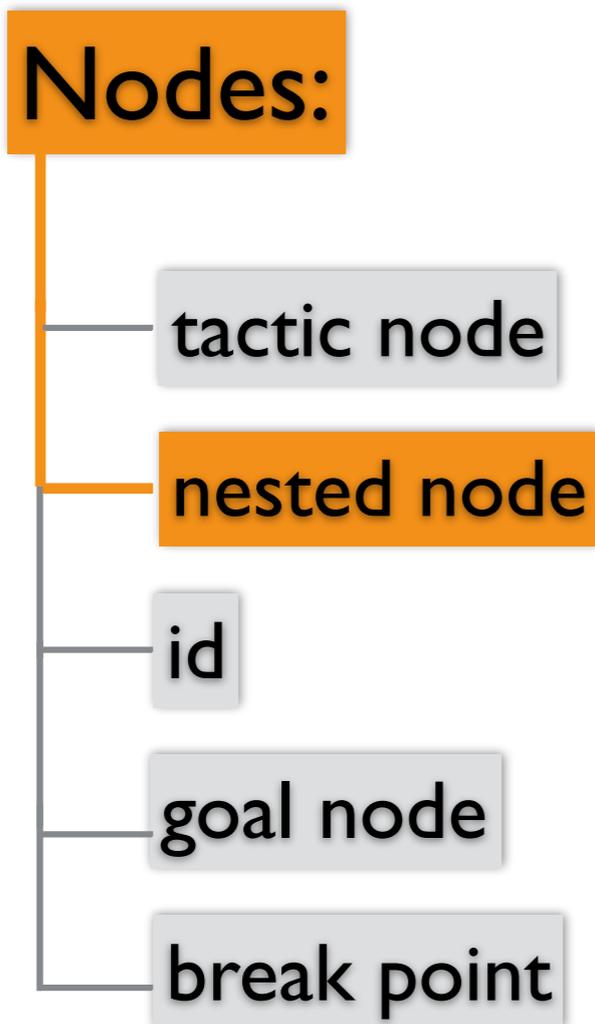
goal node

break point

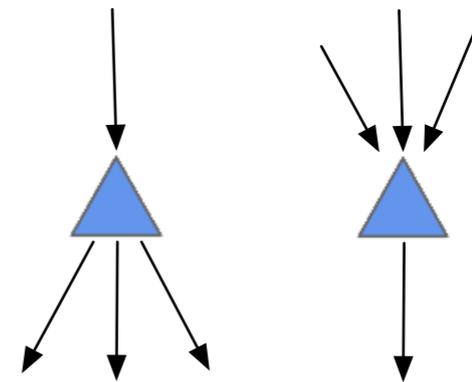
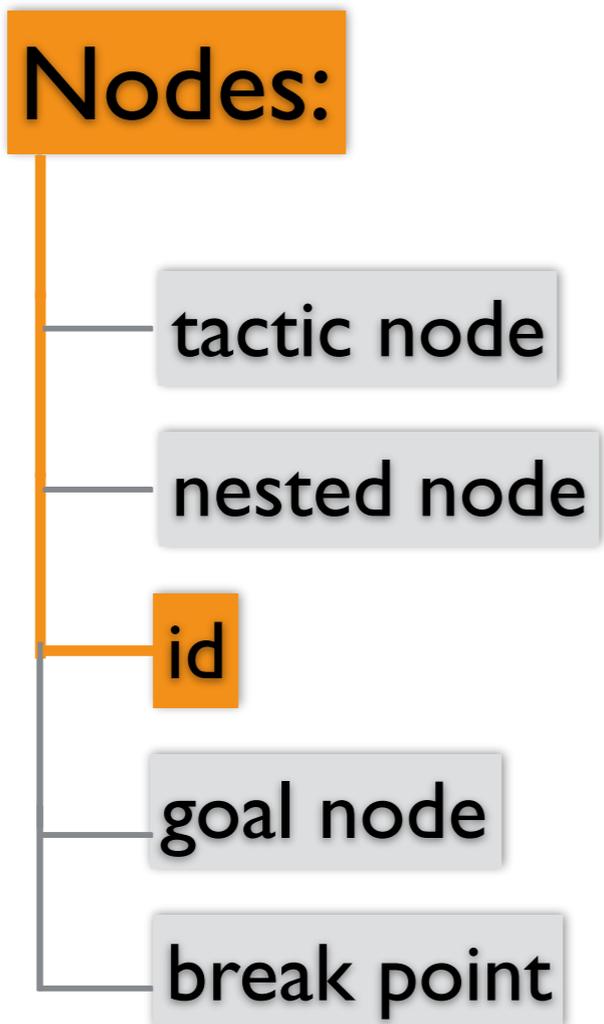
# Tinker Tool



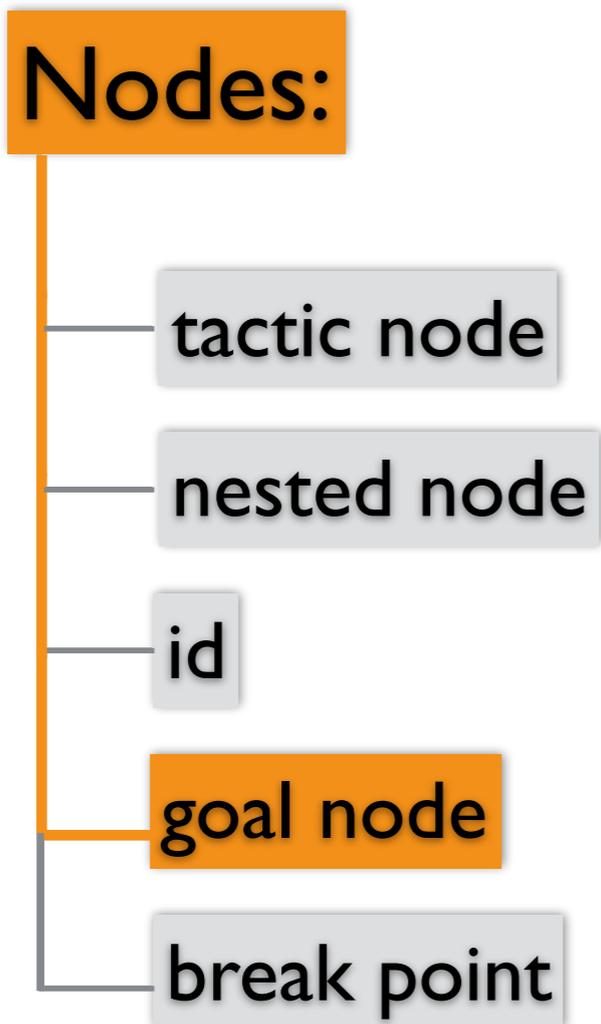
# Tinker Tool



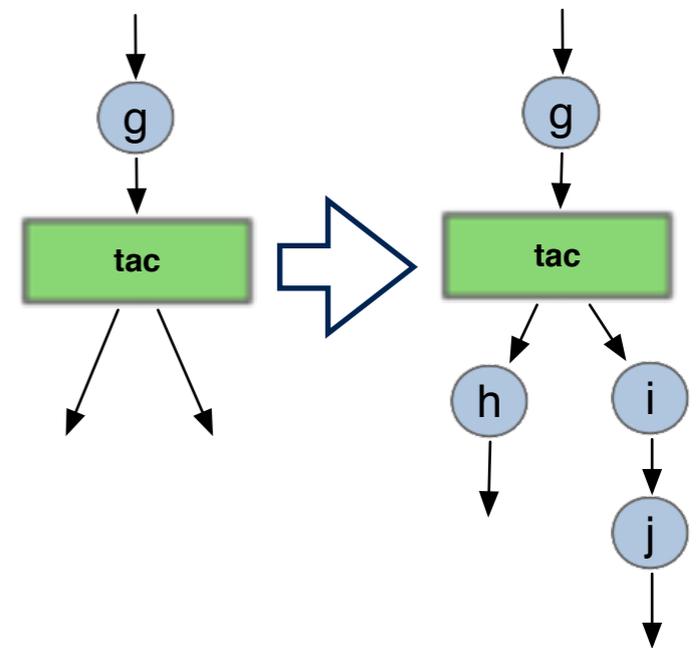
# Tinker Tool



# Tinker Tool



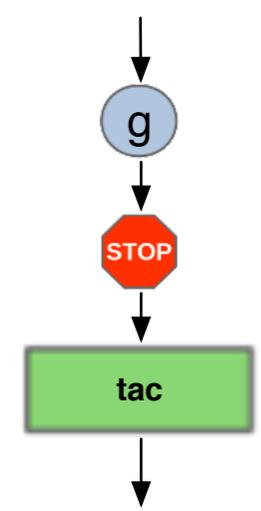
$\text{tac}(g)=[h,i,j]$



# Tinker Tool

## Nodes:

- tactic node
- nested node
- id
- goal node
- break point

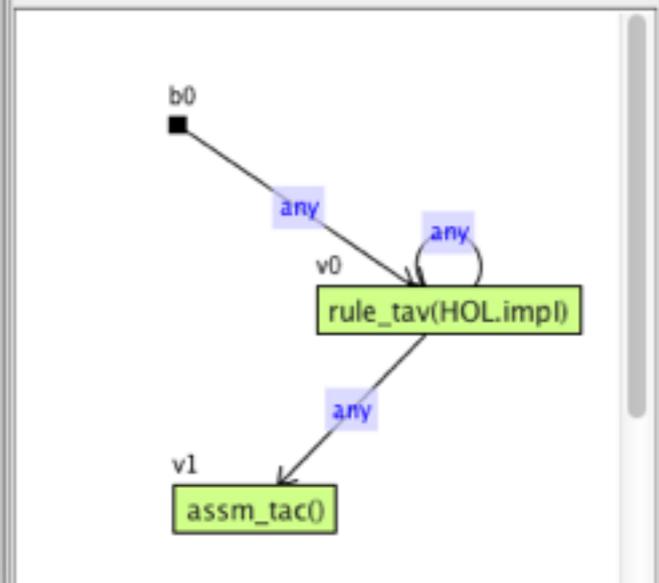


### Library

- tinker\_library
  - demo
    - demo.psgraph
    - demo\_library.psgraph
  - dev
    - lemma\_tac.psgraph
  - rodin
    - TA.psgraph
    - demo\_rodin.psgraph
    - oddEven.psgraph

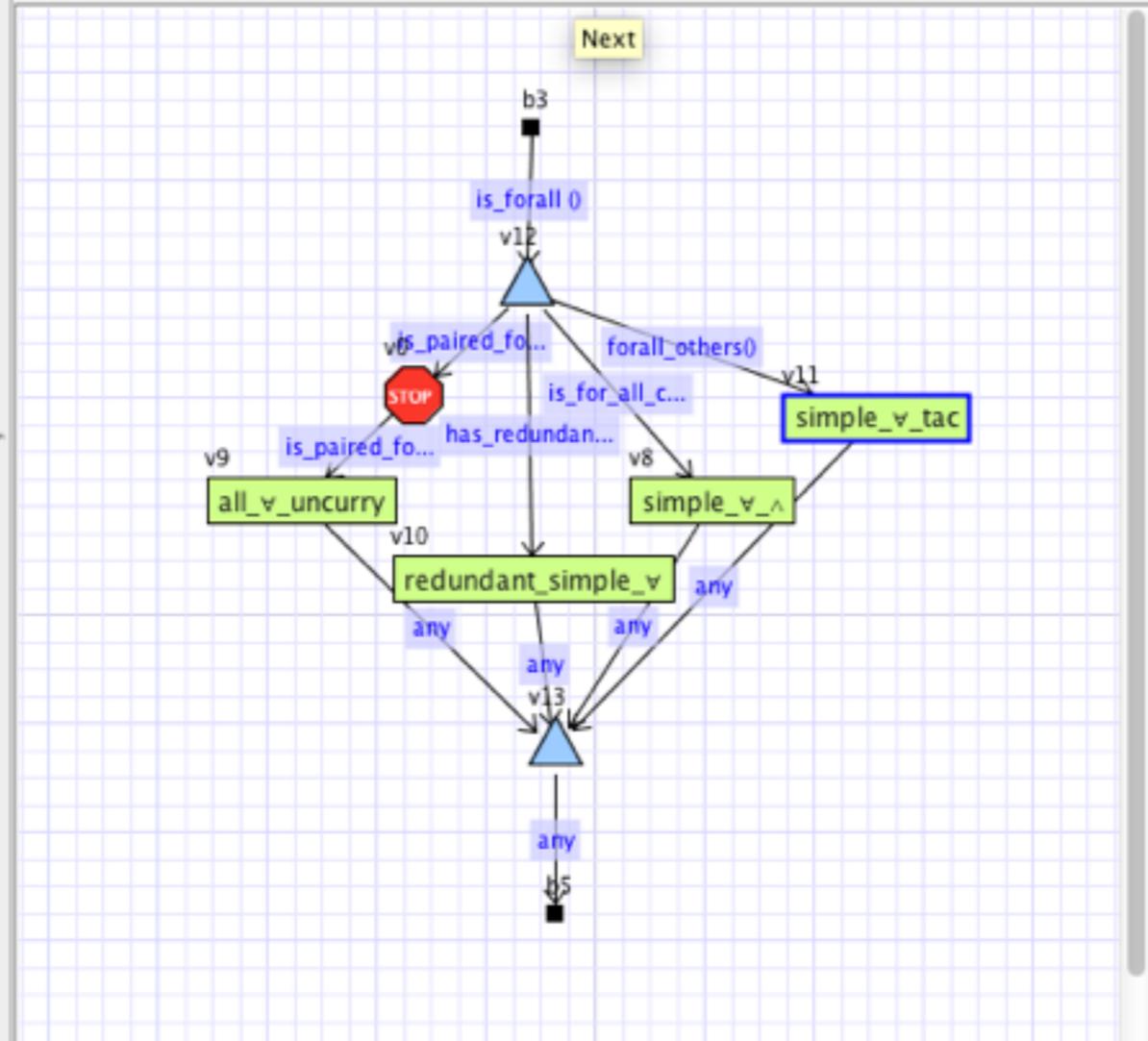
### demo\_library.psgraph preview

demo .ry



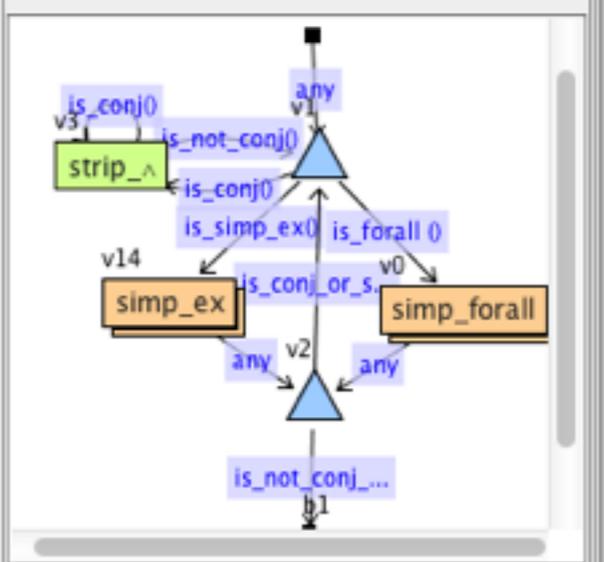
simple\_quantifier\_tac ▶ simp\_foral 1 / 1

Navigation icons: back, forward, search, zoom, etc.



### Tactic inspector

simple\_quantifier\_tac



### Node Information

Node : v11

Type : Atomic tactic

Name : simple\_v\_tac

Value :

✎ ✖



Tinker - simple\_quantifier\_tac.psgraph

File Edit Debug Record

**Library**

- tinker\_library
  - demo
    - demo.psgraph
    - demo\_library.psgraph
  - dev
    - lemma\_tac.psgraph
  - rodin
    - TA.psgraph
    - demo\_rodin.psgraph
    - oddEven.psgraph

**demo\_library.psgraph preview**

demo .ry

**simple\_quantifier\_tac** ▶ simp\_foral 1 / 1

**Graph view**

**Tactic inspector**

simple\_quantifier\_tac

**Node Information**

Node : v11

Type : Atomic tactic

Name : simple\_v\_tac

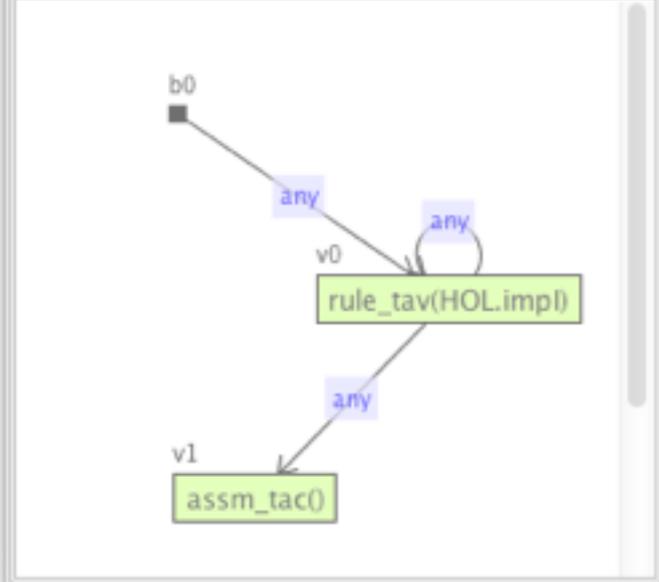
Value :

**Library**

- tinker\_library
  - demo
    - demo.psgraph
    - demo\_library.psgraph
  - dev
    - lemma\_tac.psgraph
  - rodin
    - TA.psgraph
    - demo\_rodin.psgraph
    - oddEven.psgraph

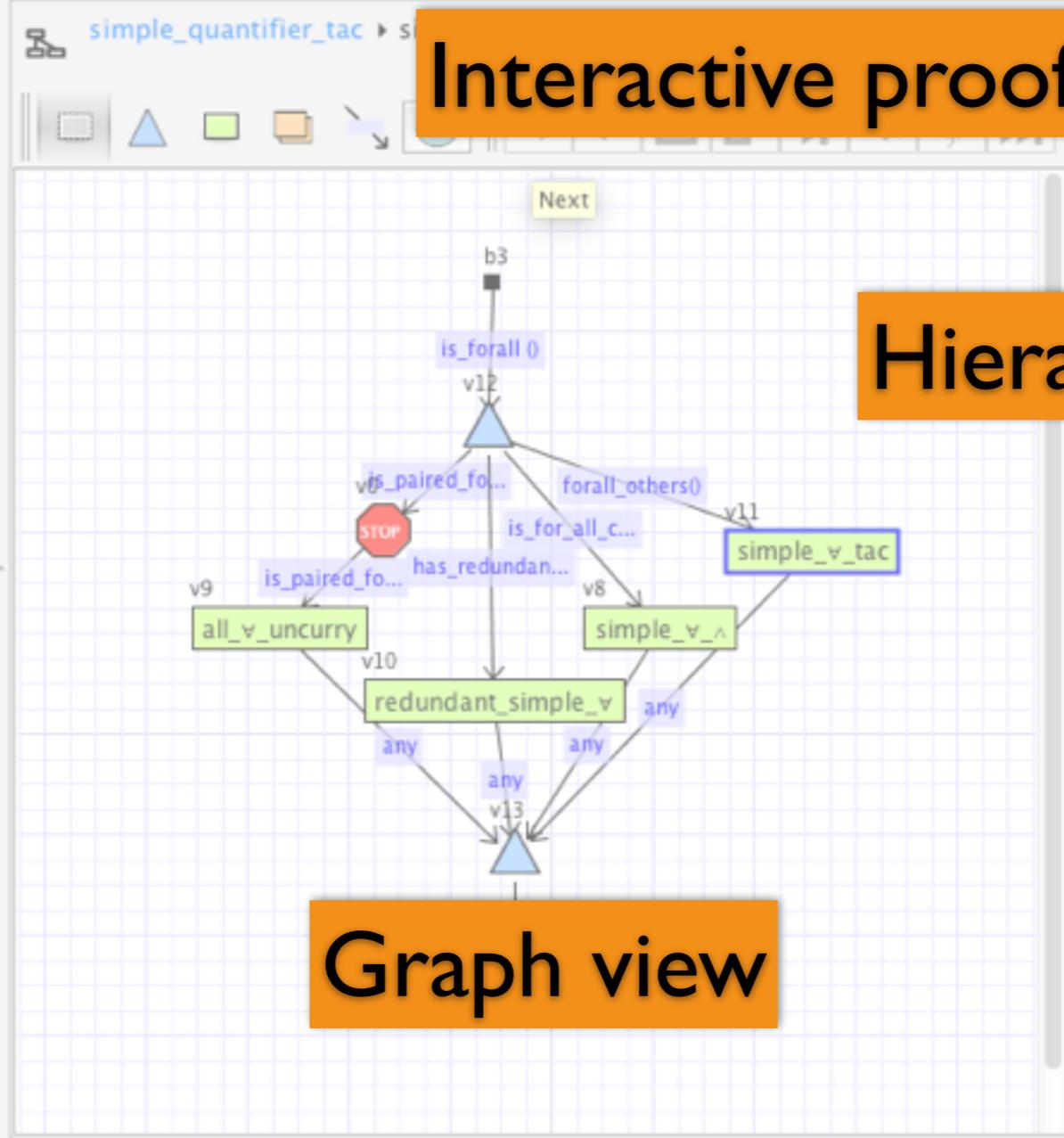
**demo\_library.psgraph preview**

demo ← .ry → 🔍 🔍 ↻

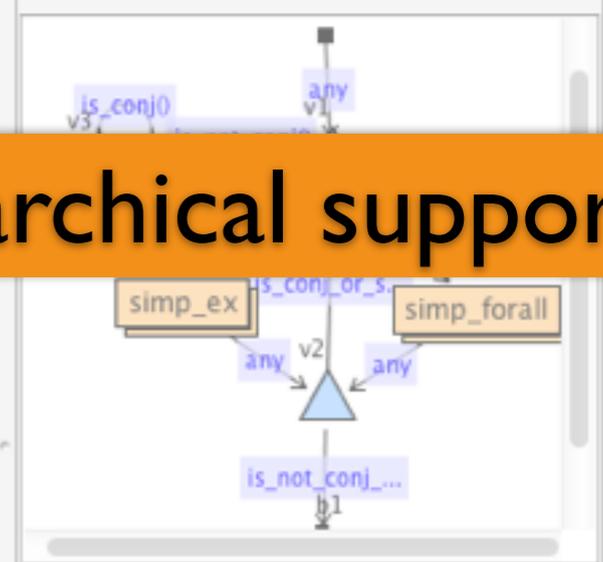


Interactive proof inspection

Hierarchical supports



Graph view



**Node Information**

Node : v11

Type : Atomic tactic

Name : simple\_v\_tac

Value :

✎ ✖

Tinker - simple\_quantifier\_tac.psgraph

File Edit Debug Record

Library

- tinker\_library
  - demo
    - demo.psgraph
    - demo\_library.psgraph
  - TA.psgraph
  - demo\_rodin.psgraph
  - oddEven.psgraph

demo\_library.psgraph preview

demo .ry

Library

Interactive proof inspection

Hierarchical supports

Graph view

```
graph TD
    b0[b0] -- any --> v0((v0))
    v0 -- any --> rule_tav[rule_tav(HOL.impl)]
    rule_tav -- any --> v1((v1))
    v1 -- any --> assm_tac[assm_tac()]
    b3[b3] --> v12((v12))
    v12 -- is_forall_0 --> v13((v13))
    v12 -- forall_others0 --> v11((v11))
    v12 -- is_for_all_c... --> v10((v10))
    v12 -- has_redundan... --> v9((v9))
    v12 -- is_paired_fo... --> v8((v8))
    v12 -- is_paired_fo... --> v7((v7))
    v9 --> all_v_uncurry[all_v_uncurry]
    v8 --> simple_v_1[simple_v_1]
    v7 --> simple_v_2[simple_v_2]
    v10 --> redundant_simple_v[redundant_simple_v]
    v11 --> simple_v_tac[simple_v_tac]
    all_v_uncurry -- any --> v13
    simple_v_1 -- any --> v13
    simple_v_2 -- any --> v13
    redundant_simple_v -- any --> v13
    simple_v_tac -- any --> v13
    v13 --> v12
```

Node Information

Node : v11

Type : Atomic tactic

Name : simple\_v\_tac

Value :

# Record & replay

# Interactive proof inspection

# Library

# Hierarchical supports

# Graph view

The screenshot displays the Tinker proof assistant interface for the file `simple_quantifier_tac.psgraph`. The interface is divided into several panels:

- Library Panel (Left):** Shows a tree view of the `tinker_library` containing a `demo` folder with files like `demo.psgraph`, `TA.psgraph`, `demo_rodin.psgraph`, and `oddEven.psgraph`. Below this is a `demo_library.psgraph preview` section with a search bar and navigation icons.
- Main Graph View (Center):** A large grid-based area showing a complex proof graph. The graph starts with a root node `b3` (black square) leading to `is_forall 0` (blue triangle) at node `v12`. From `v12`, three paths emerge: one to a red `STOP` node via `is_paired_fo...` (node `v11`), one to `forall_others 0` (node `v8`), and one to `is_for_all_c...` (node `v10`). Node `v10` leads to `all_v_uncurry` (green rectangle) at node `v9`. Node `v8` leads to `simple_v_∧` (green rectangle) at node `v7`. Node `v11` leads to `simple_v_tac` (blue rectangle) at node `v6`. Node `v7` leads to `redundant_simple_v` (green rectangle) at node `v5`. Node `v6` leads to `redundant_simple_v` at node `v4`. Node `v5` leads to `any` (blue triangle) at node `v3`. Node `v4` leads to `any` at node `v2`. Node `v3` leads to `any` at node `v1`. A `Next` button is visible at the top right of the graph area.
- Node Information Panel (Right):** A panel titled "Node Information" showing details for node `v11`:
  - Node : v11
  - Type : Atomic tactic
  - Name : simple\_v\_tac
  - Value :Below the text are icons for editing (pencil) and deleting (red X).

Tinker - simple\_quantifier\_tac.psgraph

File Edit Debug Record

Library

- tinker\_library
  - demo
    - demo.psgraph
    - demo\_library.psgraph
  - dev
    - lemma\_tac.psgraph
  - rodin
    - TA.psgraph
    - demo\_rodin.psgraph
    - oddEven.psgraph

demo\_library.psgraph preview

demo .ry

rule\_tav(HOL.impl)

assm\_tac()

simple\_quantifier\_tac ▶ simp\_foral 1 / 1

Next

# Tool Demo

Tactic inspector

simple\_quantifier\_tac

Node Information

Node : v11

Type : Atomic tactic

Name : simple\_v\_tac

Value :

<http://ggrov.github.io/tinker/>