



Atomated Theory Substitution

Toward Proof-Driven Software Development

HCSS May 6, 2024
John Scebold, Jared Ziegler

[TWOSIXTECH.COM](https://twosixtech.com)

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

TOC

- Introduction
- Test-Driven Development
- Proof-Driven Development
- AMETHYST AI
- AMETHYST UX
- Future

AMETHYST Introduction

- Formal verification of real world software has shown increasing success
 - seL4
 - CompCert
- But still no widespread adoption
 - Expert knowledge required to write and maintain formal proofs
 - Experience with interactive theorem provers (ITP)
- AutoMatEd THeorY SubsTitution (AMETHYST)
 - Artificially intelligent proof assistant for the Isabelle ITP
 - User interaction designed for non-experts
 - Software engineers
 - Integrated into VSCode
 - Move maintenance from the expert into the CI/CD cycle

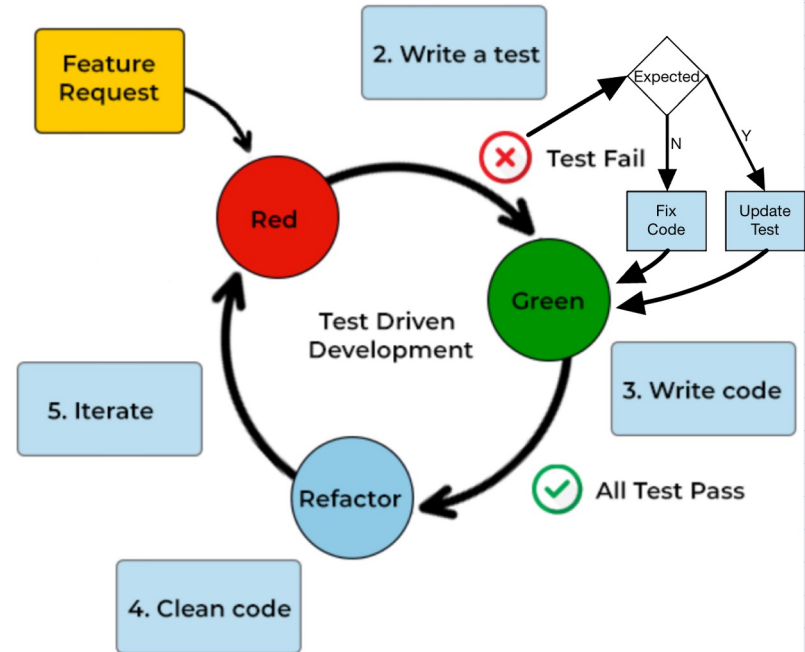
Empower the novice, Unburden the expert

Distribution Statement A. Approved for public release: distribution is unlimited.

Test-Driven Development

1. Start here

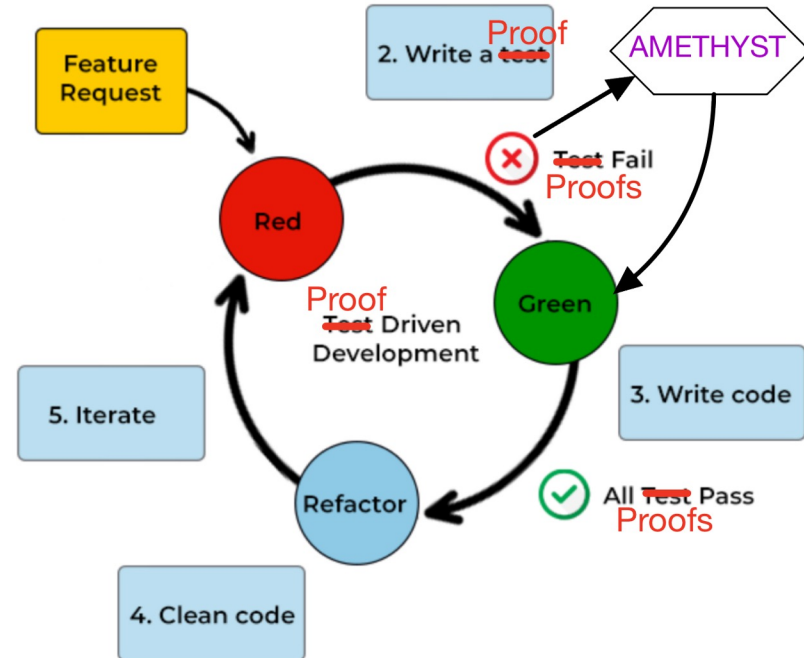
- Tests may be
 - Happy path scenarios
 - Edge case scenarios
 - Regression/unit
- Often run automatically before commit
- Failed tests can go in two directions
 - New bug in code - fix it
 - Expected change in code - update test
- Pro: Tests are written in code so the developer can maintain code and tests
- Con: Can never explicitly cover all cases



Proof-Driven Development

1. Start here

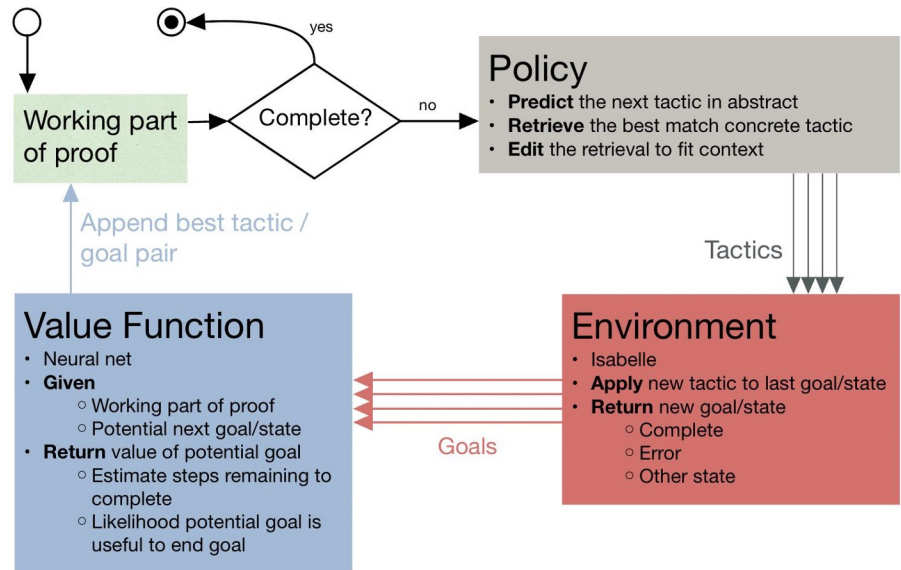
- Proofs implicitly cover all cases
- Should run automatically before commit
- It is easier to break a test than to break a proof
- Fixing a proof is less straightforward - may be a combination of code and proof edits
- Code is easy, proofs are hard
- Amethyst can help with the proof



AMETHYST AI

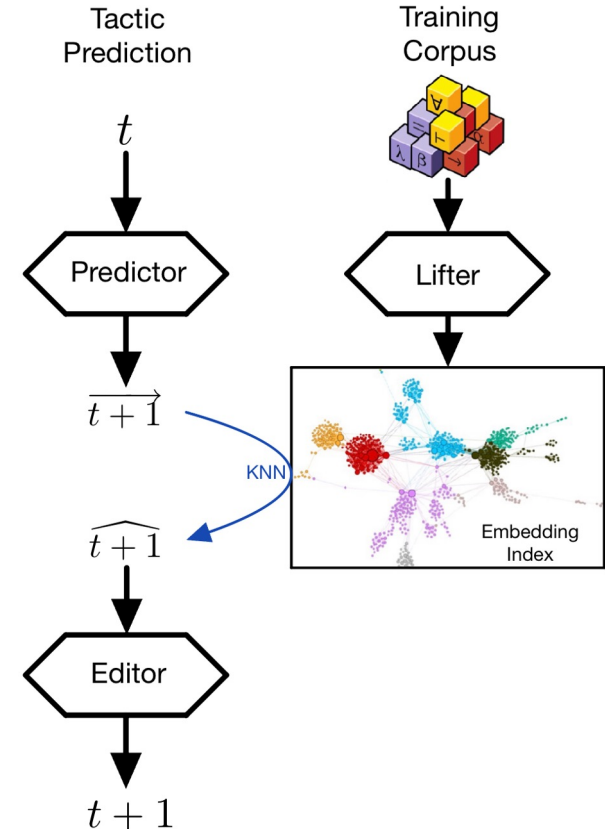
Amethyst AI

- Two machine learning models
 - **Policy** and **Value Function**
 - Pretrained offline on seL4 and Isabelle proofs
- Isabelle/HOL proof assistant
- Coordinated by a tree search agent
 - Finishes a partial proof
 - Updates models in online learning fashion



Policy

- Lifter
 - Pretrained to embed lexically similar tactics near each other
 - Embeddings are indexed for NN lookup
- Predictor
 - Given text of a tactic, predict embedding of next tactic
 - Lookup NN in index
- Editor
 - GPT edits the retrieval to fit context of current proof
 - This is remedial RAG

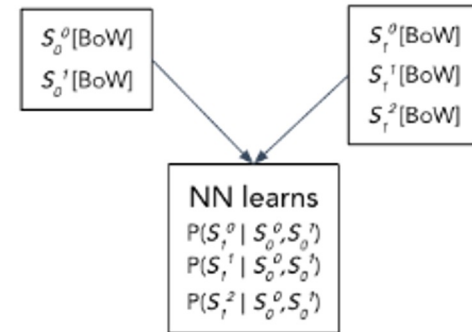


Value Function

- Proof: series of interleaved states and tactics
 - A state s is composed of one or more sub goals
 - Tactic t_n operates on s_n to produce s_{n+1}
- Measure: given states 0-n, how useful is s_{n+1} to completing the proof?
- Train a neural network to predict if the sub goals of s_{n+1} truly follow sub goals of s_n
 - Positive example - select s_n, s_{n+1} from contiguous proofs
 - Negative example - select s_n from one proof, select s_{n+1} from random other proof

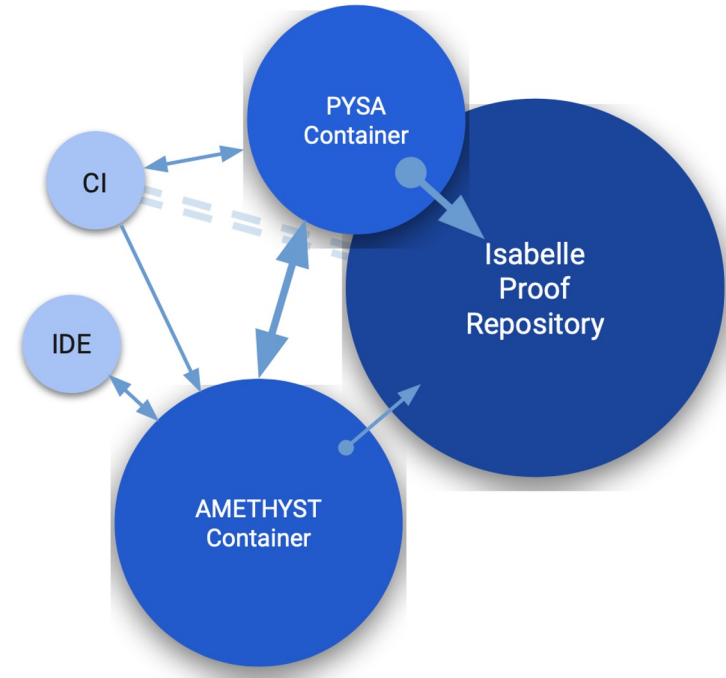
S_0 • proof (prove) goal (1 subgoal): • 1. preList xs SKIP s = postList xs s	S_1 • proof (prove) goal (2 subgoals): • 1. preList [] SKIP s = postList [] s • 2. $\forall \langle \text{And} \rangle a \text{ xs. preList xs SKIP s = postList xs s} \rightarrow$ $\text{preList (a \# xs) SKIP s = postList (a \# xs) s}$
--	--

Encode -> Bag of Words



PYSA

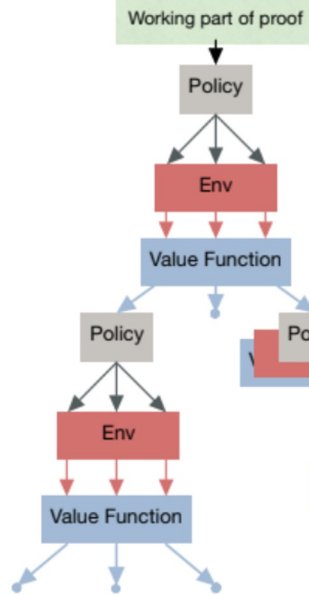
- Provide Python API for Isabelle actions
 - Inspired by Portal to Isabelle (PISA)
 - Parse and navigate theory files
 - Build heap images
 - Gather curriculum data from git history
- Get proof state from Isabelle
 - Lesson plans for AMETHYST training
 - Feedback for AMETHYST evaluation
 - Proof code testing for CI



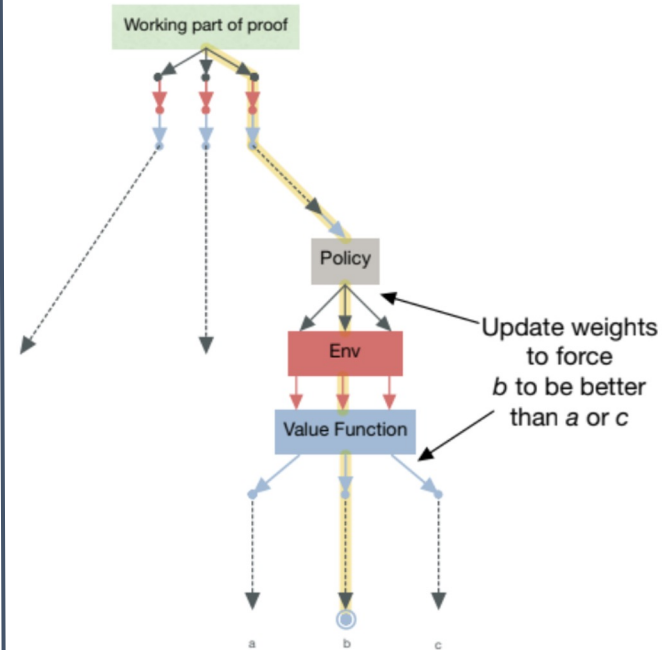
Tree Search Agent

Agent "owns" all modules

- Receives partial working proof
- Policy predicts possible next tactics
- Execute tactics via PYSA
- Value function scores valid PYSA returns
- Iterate until complete proof or max depth
- Take user feedback and back propagate up the tree from best leaf updating models.

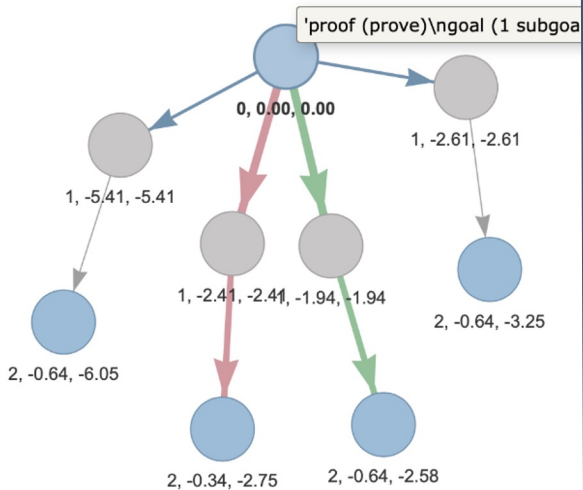


Agent uses Best-first search or MCTS to explore best value node whatever level of the tree it may be on at that step.



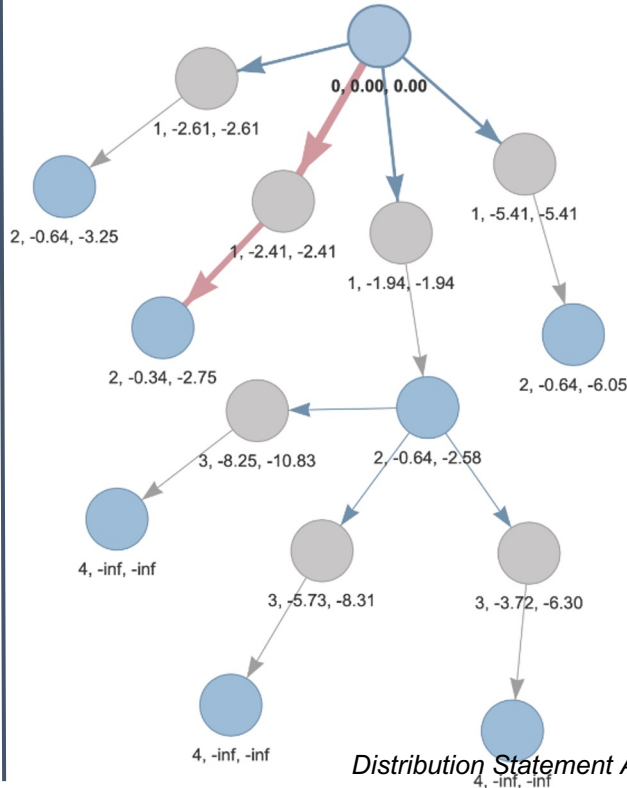
Tree Search Teacher-Forced Learning

Step 1



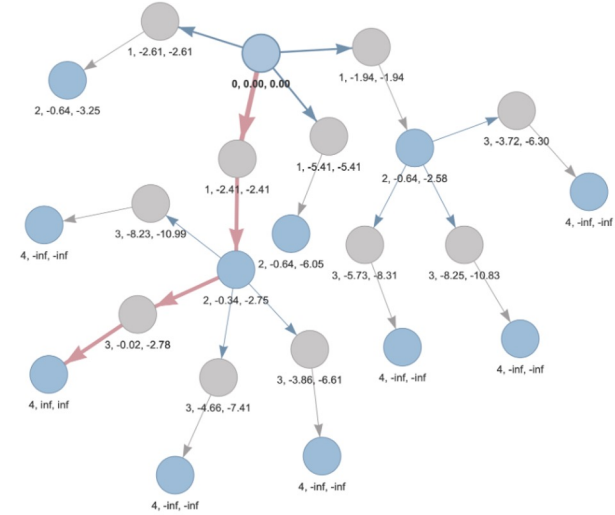
- Blue node - proof state
- Grey node - generated tactic
- Green edge - best path
- Red Edge - teacher's correct path

Step 2



Distribution Statement A.

Step 3



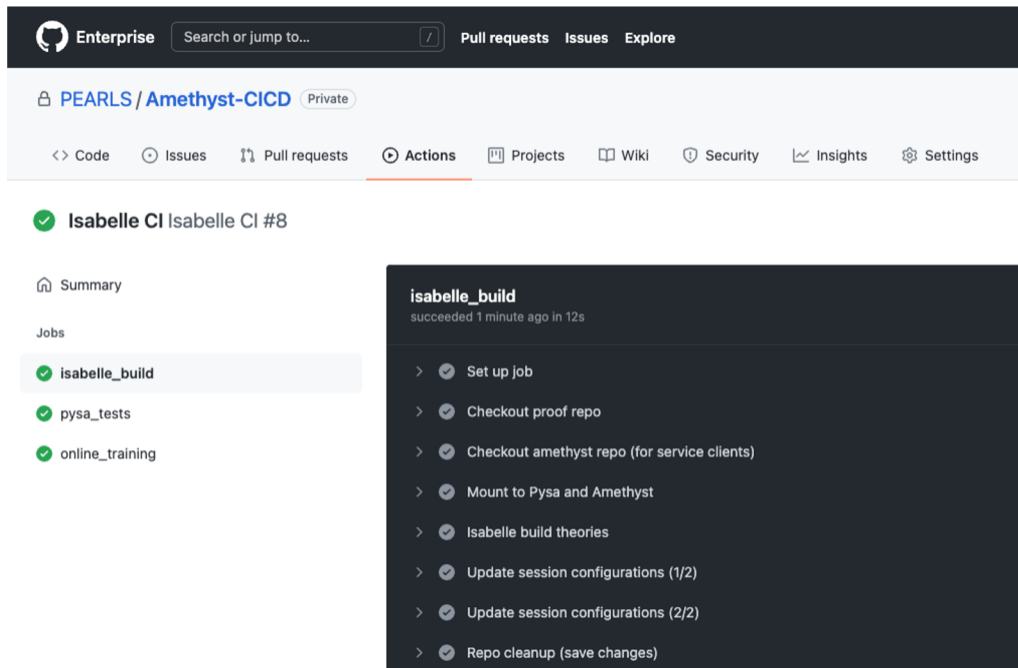
- Search terminates when PYSA returns zero subgoals.
- Value function assigns score of infinity.
- Teacher path is highest scoring.

Approved for public release: distribution is unlimited.

AMETHYST UX

CI with GitHub Actions

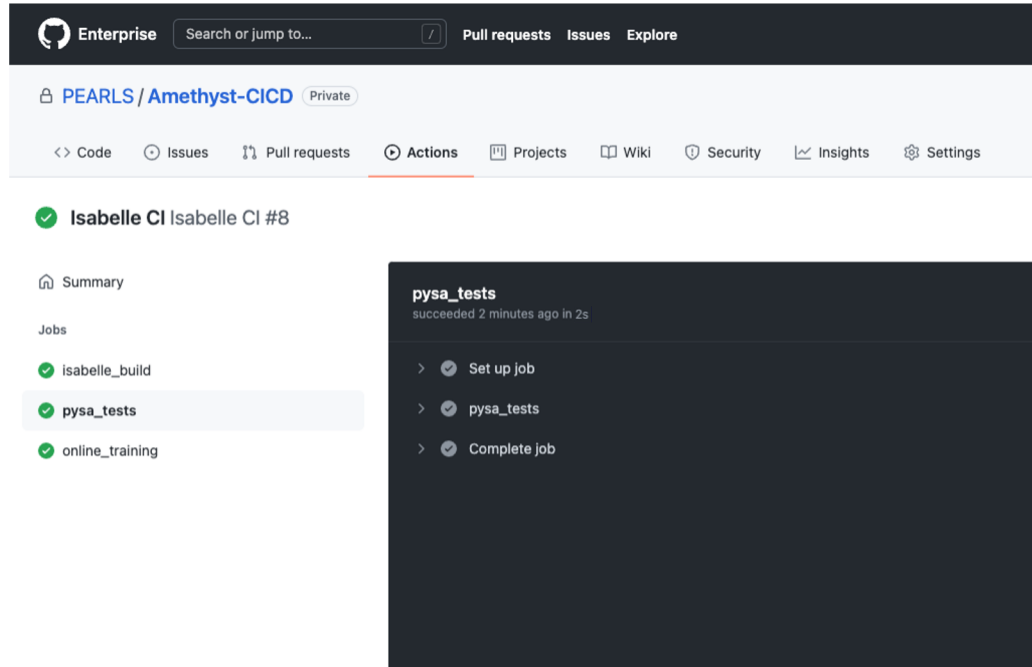
- Builds modified theories, reports meaningful deltas



The screenshot shows the GitHub Actions interface for a workflow named 'Isabelle CI' in the repository 'PEARLS / Amethyst-CICD'. The workflow run is titled 'Isabelle CI Isabelle CI #8' and is in a successful state. The 'Jobs' section lists three jobs: 'isabelle_build', 'pysa_tests', and 'online_training', all of which are completed successfully. A detailed view of the 'isabelle_build' job is shown, indicating it succeeded 1 minute ago in 12 seconds. The job steps include: 'Set up job', 'Checkout proof_repo', 'Checkout amethyst repo (for service clients)', 'Mount to Pysa and Amethyst', 'Isabelle build theories', 'Update session configurations (1/2)', 'Update session configurations (2/2)', and 'Repo cleanup (save changes)'.

CI with GitHub Actions

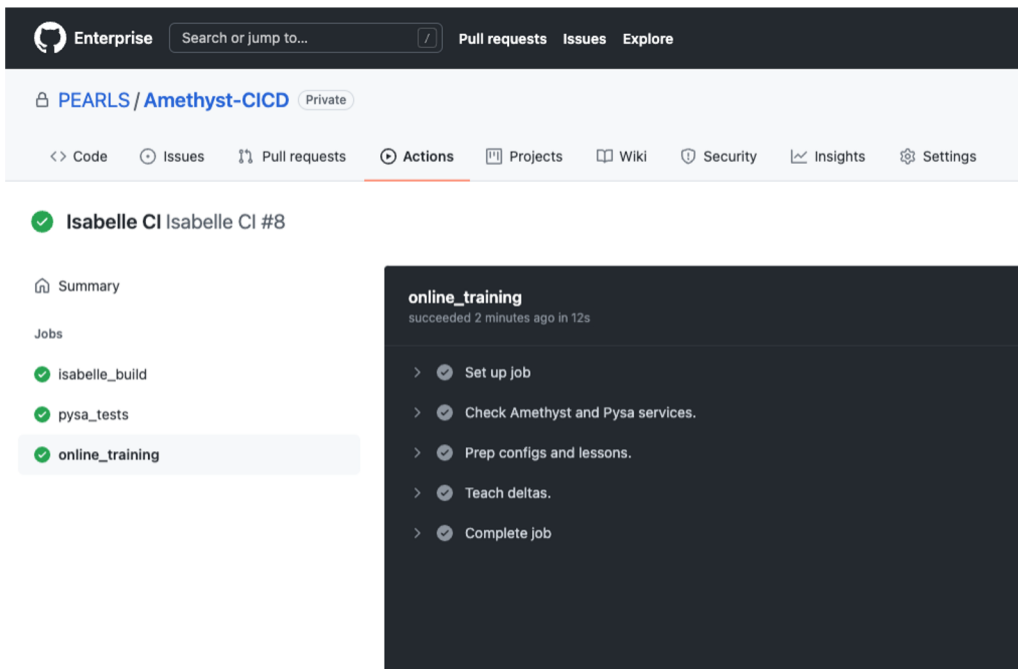
- Builds modified theories, reports meaningful deltas
- Tests all proofs, checking PYSA capabilities.



The screenshot shows the GitHub Actions interface for a workflow named 'Isabelle CI' in the repository 'PEARLS / Amethyst-CICD'. The workflow run is titled 'Isabelle CI Isabelle CI #8' and is in a successful state, indicated by a green checkmark. The 'Jobs' section lists three jobs: 'isabelle_build', 'pysa_tests', and 'online_training'. The 'pysa_tests' job is selected and expanded, showing a detailed view of its execution. The job 'pysa_tests' succeeded 2 minutes ago. The job steps are: 'Set up job', 'pysa_tests', and 'Complete job', all of which are marked as successful with green checkmarks.

CI with GitHub Actions

- Builds modified theories, reports meaningful deltas
- Tests all proofs, checking Pysa capabilities.
- Online training: teaches model new proofs automatically

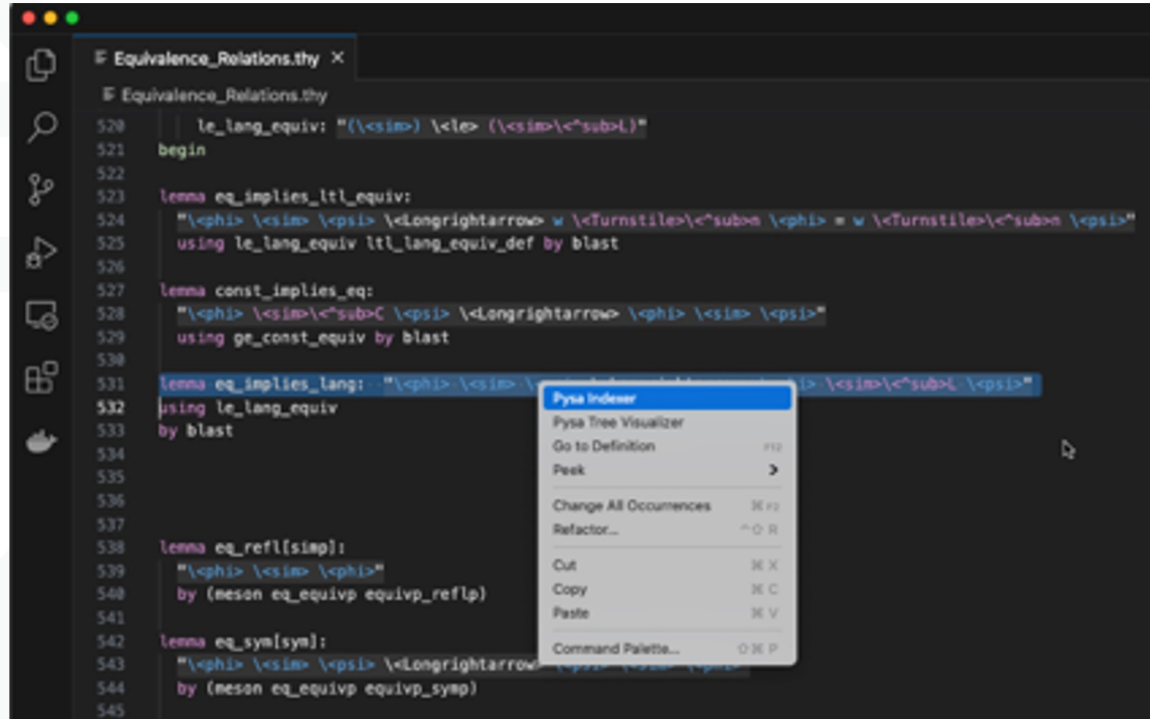


The screenshot shows the GitHub Actions interface for a workflow named 'Isabelle CI' in the repository 'PEARLS / Amethyst-CICD'. The workflow run is titled 'Isabelle CI Isabelle CI #8' and is in a 'Completed' state, indicated by a green checkmark. The workflow summary shows three jobs: 'isabelle_build', 'pysa_tests', and 'online_training', all of which are completed. The 'online_training' job is expanded to show its steps: 'Set up job', 'Check Amethyst and Pysa services.', 'Prep configs and lessons.', 'Teach deltas.', and 'Complete job', all of which are also completed.

VSCoDe Extension

- Indexer
 - Search tool based on KNN in an embedding space
 - Helps find similar lemmas or tactics
- Interactive Proof Search
 - Visualize AI driven proof search
 - See which branches are promising
 - Proof states ranked by a score
 - Direct agent to explore certain paths

Indexer

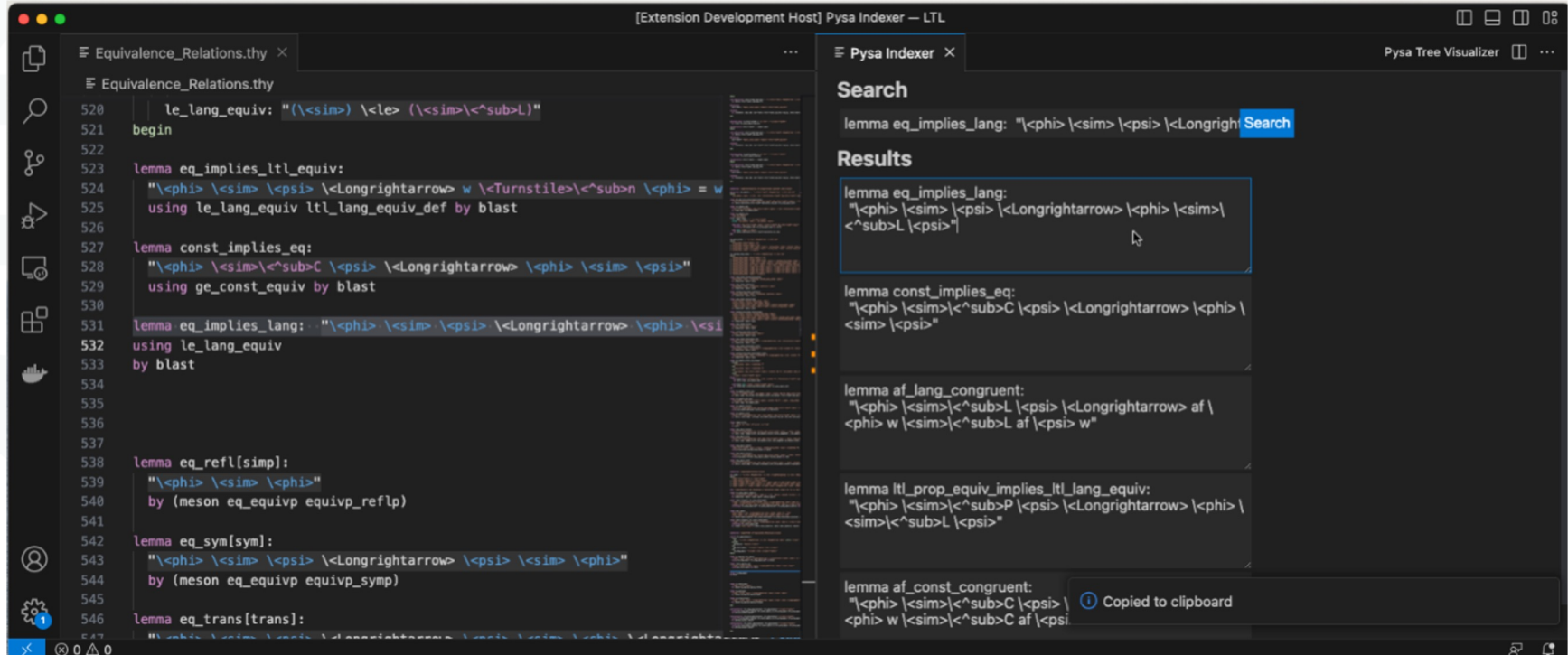


```

Equivalence_Relations.thy x
Equivalence_Relations.thy
520   le_lang_equiv: "{(\<sim>) \<le> (\<sim>\<sub>L})"
521   begin
522
523   lemma eq_implies_ltl_equiv:
524     "{\<phi> \<sim> \<psi> \<longrightarrow> w \<Turnstile>\<sub>w \<phi> = w \<Turnstile>\<sub>w \<psi>"
525     using le_lang_equiv ltl_lang_equiv_def by blast
526
527   lemma const_implies_eq:
528     "{\<phi> \<sim>\<sub>C \<psi> \<longrightarrow> \<phi> \<sim> \<psi>"
529     using ge_const_equiv by blast
530
531   lemma eq_implies_lang: "{\<phi> \<sim> \<psi> \<longrightarrow> \<phi> \<sim>\<sub>C \<psi>}"
532   using le_lang_equiv
533   by blast
534
535
536
537
538   lemma eq_refl[simp]:
539     "{\<phi> \<sim> \<phi>}"
540     by (meson eq_equivp equivp_refl)
541
542   lemma eq_sym[sym]:
543     "{\<phi> \<sim> \<psi> \<longrightarrow> \<psi> \<sim> \<phi>}"
544     by (meson eq_equivp equivp_sym)
545
  
```

The screenshot shows a code editor window titled "Equivalence_Relations.thy". The code defines several lemmas related to equivalence relations. A context menu is open over the lemma definition on line 532, listing actions such as "Pyna Indexer", "Pyna Tree Visualizer", "Go to Definition", "Peek", "Change All Occurrences", "Refactor...", "Cut", "Copy", "Paste", and "Command Palette...".

Indexer



The screenshot displays the Pysa Indexer interface, which is used for indexing and searching through mathematical proof scripts. The interface is split into two main panes.

Left Pane (Code Editor): Shows the source code for `Equivalence_Relations.thy`. The code defines several lemmas related to equivalence relations and congruence. Key lemmas include:

- `lemma eq_implies_ltl_equiv:` which states that if $\phi \sim \psi$, then $\phi \sim \psi$ is preserved under the LTL operator L .
- `lemma const_implies_eq:` which states that if $\phi \sim \psi$, then $\phi \sim \psi$ is preserved under the constant operator C .
- `lemma eq_implies_lang:` which states that if $\phi \sim \psi$, then $\phi \sim \psi$ is preserved under the language operator Lang .
- `lemma eq_refl[simp]:` which states that $\phi \sim \phi$.
- `lemma eq_sym[sym]:` which states that if $\phi \sim \psi$, then $\psi \sim \phi$.
- `lemma eq_trans[trans]:` which states that if $\phi \sim \psi$ and $\psi \sim \chi$, then $\phi \sim \chi$.

Right Pane (Search Results): Shows the results of a search for the lemma `lemma eq_implies_lang`. The search results are displayed in a list format, with the first result highlighted in a blue box:

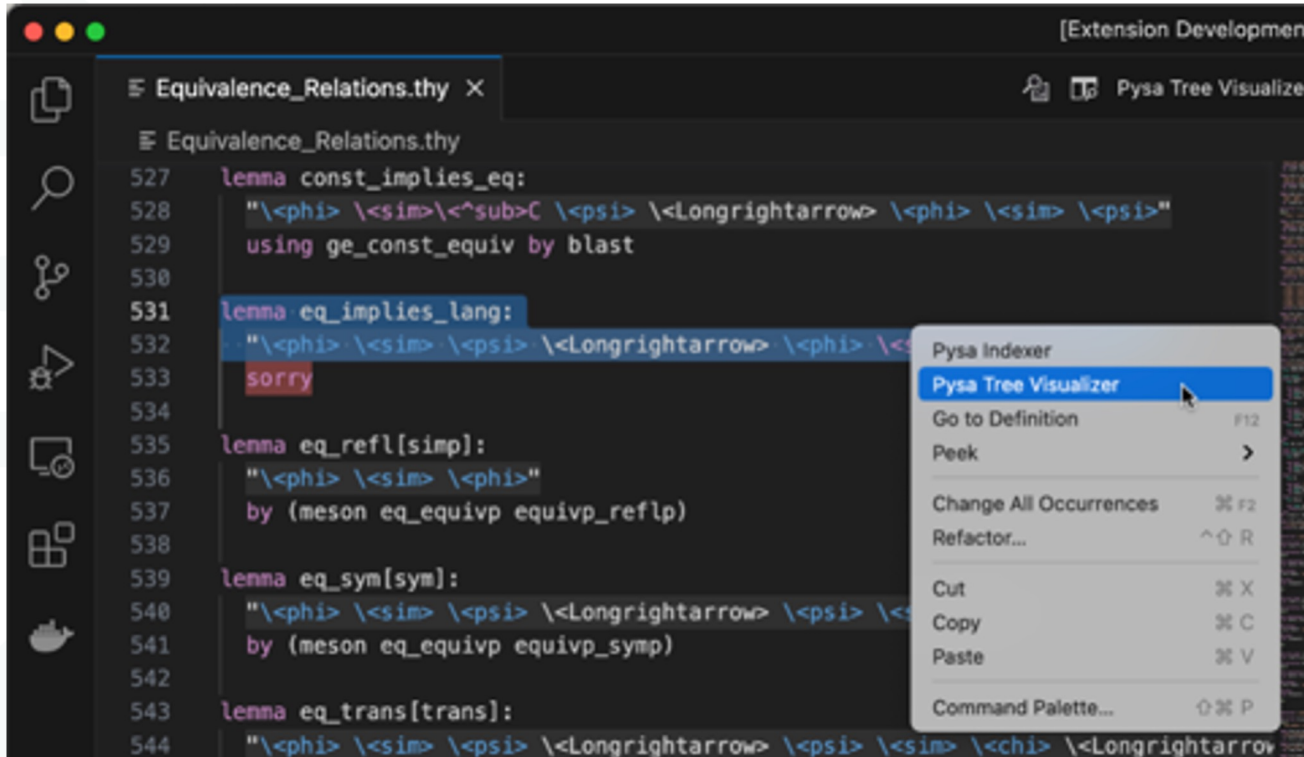
```
lemma eq_implies_lang:
  "\phi \sim \psi \Longrightarrow \phi \sim \psi"
  <sub>L</sub> \psi"
```

Below the search results, there is a notification that the selected result has been copied to the clipboard.

VSCoDe Extension

- Indexer
 - Search tool based on KNN in an embedding space
 - Helps find similar lemmas or tactics
- Interactive Proof Search
 - Visualize AI driven proof search
 - See which branches are promising
 - Proof states ranked by a score
 - Direct agent to explore certain paths

Proof Search Visualizer



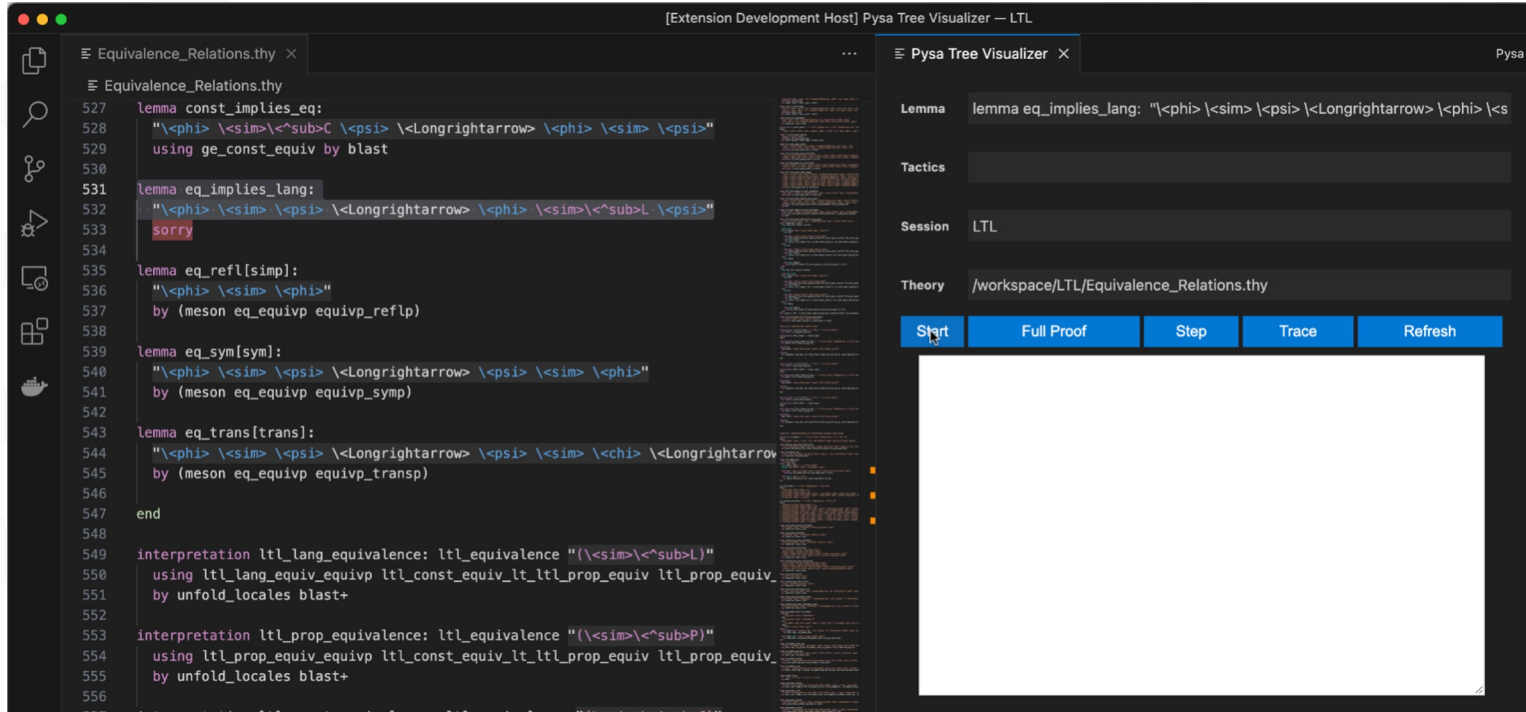
```

527 lemma const_implies_eq:
528   "\<phi> \<sim>\<sup>C \<psi> \<Longrightarrow> \<phi> \<sim> \<psi>"
529   using ge_const_equiv by blast
530
531 lemma eq_implies_lang:
532   "\<phi> \<sim> \<psi> \<Longrightarrow> \<phi> \<sim> \<psi>"
533   sorry
534
535 lemma eq_refl[simp]:
536   "\<phi> \<sim> \<phi>"
537   by (meson eq_equivp equivp_refl)
538
539 lemma eq_sym[sym]:
540   "\<phi> \<sim> \<psi> \<Longrightarrow> \<psi> \<sim> \<phi>"
541   by (meson eq_equivp equivp_sym)
542
543 lemma eq_trans[trans]:
544   "\<phi> \<sim> \<psi> \<Longrightarrow> \<psi> \<sim> \<chi> \<Longrightarrow> \<phi> \<sim> \<chi>"

```

The screenshot shows a code editor window titled "Equivalence_Relations.thy". The code contains several lemmas for equivalence relations. A context menu is open over line 532, with "Pysa Tree Visualizer" selected. Other menu items include "Pysa Indexer", "Go to Definition", "Peek", "Change All Occurrences", "Refactor...", "Cut", "Copy", "Paste", and "Command Palette...".

Proof Search Visualizer



The screenshot displays the Pysa Tree Visualizer interface. On the left, a code editor shows a theorem prover script with the following content:

```

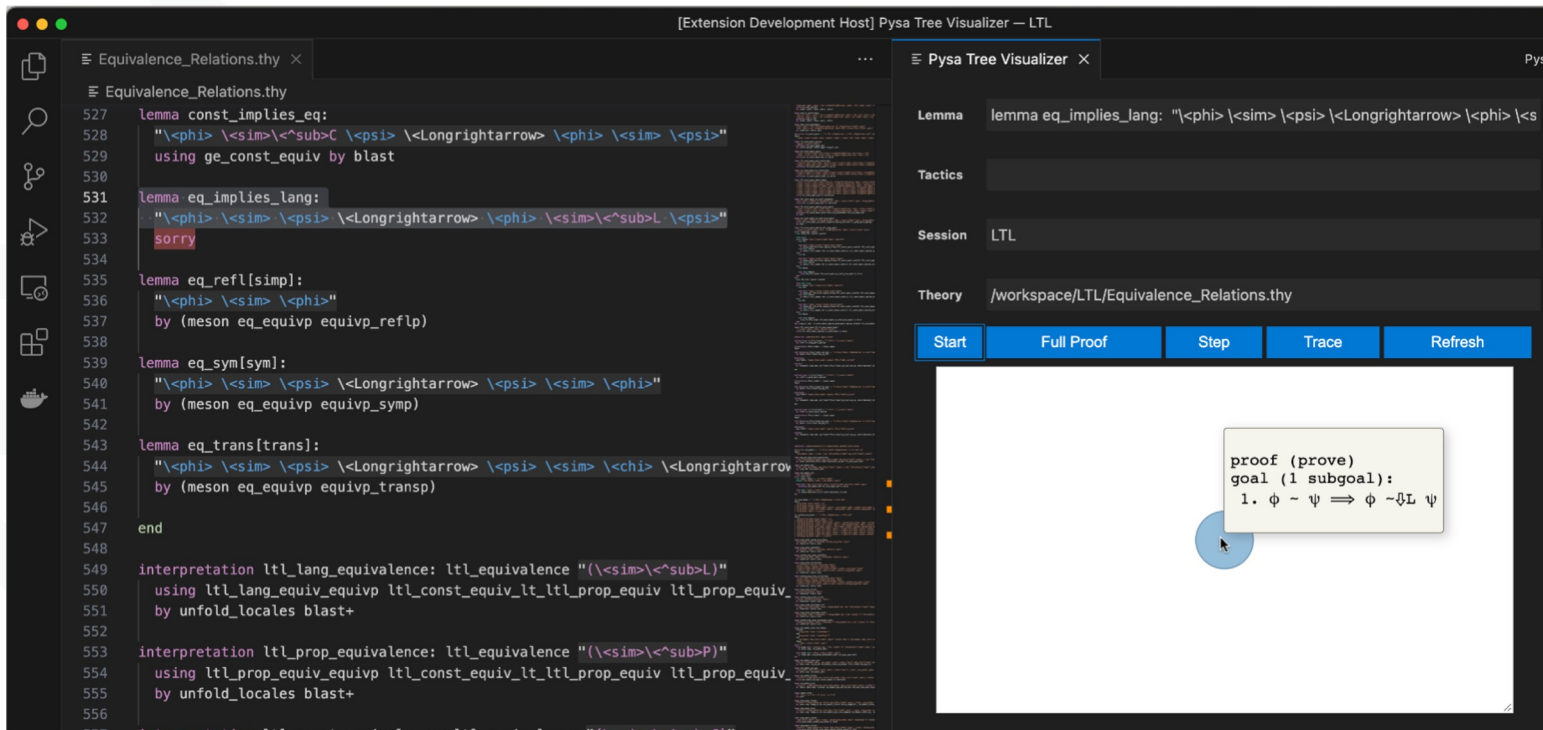
527 lemma const_implies_eq:
528   "\<phi> \<sim>\<sup>C \<psi> \<Longrightarrow> \<phi> \<sim> \<psi>"
529   using ge_const_equiv by blast
530
531 lemma eq_implies_lang:
532   "\<phi> \<sim> \<psi> \<Longrightarrow> \<phi> \<sim>\<sup>L \<psi>"
533   sorry
534
535 lemma eq_refl[simp]:
536   "\<phi> \<sim> \<phi>"
537   by (meson eq_equivp equivp_refl)
538
539 lemma eq_sym[sym]:
540   "\<phi> \<sim> \<psi> \<Longrightarrow> \<psi> \<sim> \<phi>"
541   by (meson eq_equivp equivp_sym)
542
543 lemma eq_trans[trans]:
544   "\<phi> \<sim> \<psi> \<Longrightarrow> \<psi> \<sim> \<chi> \<Longrightarrow>
545   by (meson eq_equivp equivp_trans)
546
547 end
548
549 interpretation ltl_lang_equivalence: ltl_equivalence "\<sim>\<sup>L"
550   using ltl_lang_equiv_equivp ltl_const_equiv_ltl_prop_equiv ltl_prop_equiv
551   by unfold_locales blast+
552
553 interpretation ltl_prop_equivalence: ltl_equivalence "\<sim>\<sup>P"
554   using ltl_prop_equiv_equivp ltl_const_equiv_ltl_prop_equiv ltl_prop_equiv
555   by unfold_locales blast+
556
  
```

On the right, the Pysa Tree Visualizer control panel is visible, showing the following details:

- Lemma:** lemma eq_implies_lang: "\<phi> \<sim> \<psi> \<Longrightarrow> \<phi> \<sim> \<psi>"
- Tactics:** (empty field)
- Session:** LTL
- Theory:** /workspace/LTL/Equivalence_Relations.thy

At the bottom of the control panel, there are five buttons: **Start**, **Full Proof**, **Step**, **Trace**, and **Refresh**. The **Start** button is currently selected.

Proof Search Visualizer



The screenshot displays the Pysa Tree Visualizer interface, which is used for visualizing proof search results. The interface is split into two main panels.

Left Panel (Code Editor): Shows the source code for the `Equivalence_Relations.thy` file. The code defines several lemmas and interpretations related to LTL (Linear Temporal Logic) equivalences. The visible code includes:

```

527 lemma const_implies_eq:
528   "\phi \sim \psi \sub C \psi \longrightarrow \phi \sim \psi"
529   using ge_const_equiv by blast
530
531 lemma eq_implies_lang:
532   "\phi \sim \psi \longrightarrow \phi \sim \sub L \psi"
533   sorry
534
535 lemma eq_refl[simp]:
536   "\phi \sim \psi"
537   by (meson eq_equiv equiv_refl)
538
539 lemma eq_sym[sym]:
540   "\phi \sim \psi \longrightarrow \psi \sim \phi"
541   by (meson eq_equiv equiv_sym)
542
543 lemma eq_trans[trans]:
544   "\phi \sim \psi \longrightarrow \psi \sim \chi \longrightarrow \phi \sim \chi"
545   by (meson eq_equiv equiv_trans)
546
547 end
548
549 interpretation ltl_lang_equivalence: ltl_equivalence "\sim \sub L"
550   using ltl_lang_equiv_equiv ltl_const_equiv_ltl_prop_equiv ltl_prop_equiv
551   by unfold_locales blast+
552
553 interpretation ltl_prop_equivalence: ltl_equivalence "\sim \sub P"
554   using ltl_prop_equiv_equiv ltl_const_equiv_ltl_prop_equiv ltl_prop_equiv
555   by unfold_locales blast+
556
557

```

Right Panel (Pysa Tree Visualizer): Shows the current state of the proof search. It includes a meta-lemma, tactics, session name, and theory path. Below these are control buttons for the proof search process.

Meta-lemma: `lemma eq_implies_lang: "\phi \sim \psi \longrightarrow \phi \sim \sub L \psi"`

Tactics: (Empty)

Session: `LTL`

Theory: `/workspace/LTL/Equivalence_Relations.thy`

Control buttons: `Start`, `Full Proof`, `Step`, `Trace`, `Refresh`

The visualization area shows the current goal and a list of subgoals:

```

proof (prove)
goal (1 subgoal):
1.  $\phi \sim \psi \implies \phi \sim \sub L \psi$ 

```

Proof Search Visualizer

```

527 lemma const_implies_eq:
528   "\<phi> \<sim>\<sub>C \<psi> \<Longrightarrow> \<phi> \<sim> \<psi>"
529   using ge_const_equiv by blast
530
531 lemma eq_implies_lang:
532   "\<phi> \<sim> \<psi> \<Longrightarrow> \<phi> \<sim>\<sub>L \<psi>"
533   sorry
534
535 lemma eq_refl[simp]:
536   "\<phi> \<sim> \<phi>"
537   by (meson eq_equiv equiv_refl)
538
539 lemma eq_sym[sym]:
540   "\<phi> \<sim> \<psi> \<Longrightarrow> \<psi> \<sim> \<phi>"
541   by (meson eq_equiv equiv_sym)
542
543 lemma eq_trans[trans]:
544   "\<phi> \<sim> \<psi> \<Longrightarrow> \<psi> \<sim> \<chi> \<Longrightarrow>
545   by (meson eq_equiv equiv_trans)
546
547 end
548
549 interpretation ltl_lang_equivalence: ltl_equivalence "\<sim>\<sub>L)"
550   using ltl_lang_equiv_equiv ltl_const_equiv_ltl_ltl_prop_equiv ltl_prop_equiv_
551   by unfold_locales blast+
552
553 interpretation ltl_prop_equivalence: ltl_equivalence "\<sim>\<sub>P)"
554   using ltl_prop_equiv_equiv ltl_const_equiv_ltl_ltl_prop_equiv ltl_prop_equiv_
555   by unfold_locales blast+
556

```

Pysa Tree Visualizer

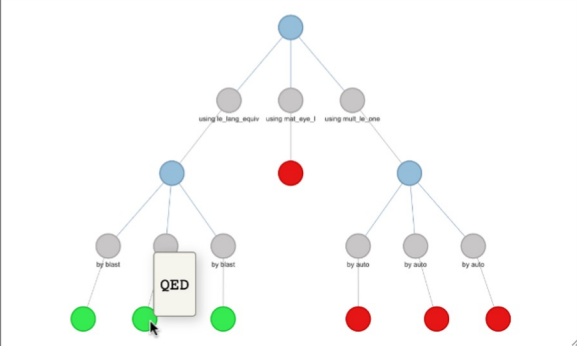
Lemma: lemma eq_implies_lang: "\<phi> \<sim> \<psi> \<Longrightarrow> \<phi> \<sim> \<psi>"

Tactics:

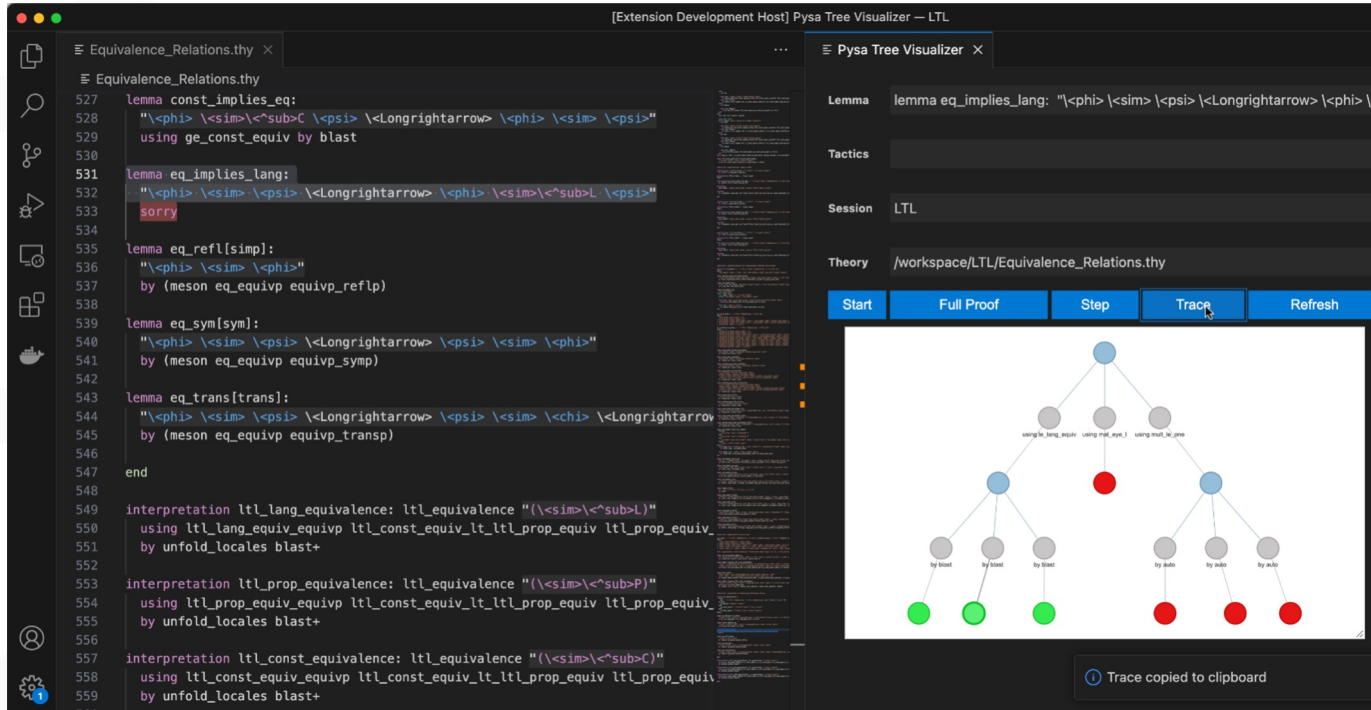
Session: LTL

Theory: /workspace/LTL/Equivalence_Relations.thy

Start
Full Proof
Step
Trace
Refresh



Proof Search Visualizer



The screenshot shows the Pysa Tree Visualizer interface. On the left, the source code for 'Equivalence_Relations.thy' is displayed, containing several lemmas and interpretations. The right pane shows the 'Pysa Tree Visualizer' window, which displays a tree diagram of the proof search process. The tree has a root node (blue) which branches into three nodes (grey). The leftmost branch leads to three leaf nodes (green), while the middle and right branches lead to red leaf nodes. A 'Trace' button is highlighted in the interface.

Proof Search Visualizer

```

527 lemma const_implies_eq:
528   "\<phi> \<sim>\<sub>C \<psi> \<Longrightrightarrow \<phi> \<sim> \<psi>"
529   using eq_const_equiv by blast
530   command "lemma"
531 lemma eq_implies_lang: "\<phi> \<sim> \<psi> \<Longrightrightarrow \<phi> \<sim>\<sub>L
532   using le_lang_equiv
533   by blast
534
535
536 lemma eq_refl[simp]:
537   "\<phi> \<sim> \<phi>"
538   by (meson eq_equiv equiv_refl)
539
540 lemma eq_sym[sym]:
541   "\<phi> \<sim> \<psi> \<Longrightrightarrow \<psi> \<sim> \<phi>"
542   by (meson eq_equiv equiv_sym)
543
544 lemma eq_trans[trans]:
545   "\<phi> \<sim> \<psi> \<Longrightrightarrow \<psi> \<sim> \<chi> \<Longrightrightarrow
546   by (meson eq_equiv equiv_trans)
547
548 end
549
550 interpretation ltl_lang_equivalence: ltl_equivalence "\<sim>\<sub>L"
551   using ltl_lang_equiv_equiv ltl_const_equiv_lt_ltl_prop_equiv ltl_prop_equiv_
552   by unfold_locales blast+
553
554 interpretation ltl_prop_equivalence: ltl_equivalence "\<sim>\<sub>P"
555   using ltl_prop_equiv_equiv ltl_const_equiv_lt_ltl_prop_equiv ltl_prop_equiv_
556   by unfold_locales blast+

```

Lemma lemma eq_implies_lang: "\<phi> \<sim> \<psi> \<Longrightrightarrow \<phi> \<sim>\<sub>L

Tactics

Session LTL

Theory /workspace/LTL/Equivalence_Relations.thy

Start
Full Proof
Step
Trace
Refresh

