Techniques and Tools for Automated Termination Analysis

Carsten Fuhs

Birkbeck, University of London

COST Action EuroProofNet, WG3 meeting, Feb 2023

Timișoara, Romania

https://www.dcs.bbk.ac.uk/~carsten/

WG3 deliverables

- Month 18: Comparison of the approaches used in the international Software Verification competition SV-COMP.
- Month 24: Software prototype for the automated inference of program specifications as logical axioms.
- Month 48: Collection of verification challenges with summary of working recipes for verifying them.
- Month 48: Technique for syntax-semantics interface for program verification with or without type systems.

WG3 deliverables

- Month 18: Comparison of the approaches used in the international Software Verification competition SV-COMP.
- Month 24: Software prototype for the automated inference of program specifications as logical axioms.
- Month 48: Collection of verification challenges with summary of working recipes for verifying them.
- Month 48: Technique for syntax-semantics interface for program verification with or without type systems.

WG3 deliverables

- Month 18: Comparison of the approaches used in the international Software Verification competition SV-COMP.
- Month 24: Software prototype for the automated inference of program specifications as logical axioms.
- Month 48: Collection of verification challenges with summary of working recipes for verifying them.
- Month 48: Technique for syntax-semantics interface for program verification with or without type systems.

⇒ Some thoughts on **competitions** and approaches for **termination**

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

- termCOMP (termination, complexity)
  https://termination-portal.org/wiki/Termination_Competition

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

- termCOMP (termination, complexity)
  https://termination-portal.org/wiki/Termination_Competition
- SV-COMP (safety, termination)
  https://sv-comp.sosy-lab.org/

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

- termCOMP (termination, complexity)
  https://termination-portal.org/wiki/Termination_Competition
- SV-COMP (safety, termination)
  https://sv-comp.sosy-lab.org/
- CoCo (confluence)
  http://project-coco.uibk.ac.at/

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

- termCOMP (termination, complexity)
  https://termination-portal.org/wiki/Termination_Competition
- SV-COMP (safety, termination)
  https://sv-comp.sosy-lab.org/
- CoCo (confluence)
  http://project-coco.uibk.ac.at/

Usually several "dimensions" for their categories, e.g.:

- Property to verify/falsify

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

- termCOMP (termination, complexity)
  https://termination-portal.org/wiki/Termination_Competition
- SV-COMP (safety, termination)
  https://sv-comp.sosy-lab.org/
- CoCo (confluence)
  http://project-coco.uibk.ac.at/

Usually several "dimensions" for their categories, e.g.:

- Property to verify/falsify
- Input language

Tool competitions on verification

Compare fully automatic verification tools on a shared benchmark set

- termCOMP (termination, complexity)
  https://termination-portal.org/wiki/Termination_Competition
- SV-COMP (safety, termination)
  https://sv-comp.sosy-lab.org/
- CoCo (confluence)
  http://project-coco.uibk.ac.at/

Usually several "dimensions" for their categories, e.g.:

- Property to verify/falsify
- Input language
- Expected output
  (claim with no proof, human-readable proof, machine-checkable proof)

Program verification: the user's perspective (1/3)

What properties of programs do we want to analyse?

Program verification: the user's perspective (1/3)

What properties of programs do we want to analyse?

- **Partial Correctness**
  $\longrightarrow$ will my program always produce the right result?

Program verification: the user's perspective (1/3)

What properties of programs do we want to analyse?

- **Partial Correctness**
  $\longrightarrow$ will my program always produce the right result?
- **Assertions by the programmer**. `assert x > 0;`
  $\longrightarrow$ will this always be true?

Program verification: the user's perspective (1/3)

What properties of programs do we want to analyse?

- **Partial Correctness**
  $\longrightarrow$ will my program always produce the right result?
- **Assertions by the programmer**. `assert x > 0;`
  $\longrightarrow$ will this always be true?
- **Memory Safety**
  $\longrightarrow$ are my memory accesses always legal?

  ```
  int* x = NULL;  *x = 42;
  ```

  $\longrightarrow$ undefined behaviour!
  $\longrightarrow$ memory safety matters: Heartbleed (OpenSSL attack)

Program verification: the user's perspective (1/3)

What properties of programs do we want to analyse?

- **Partial Correctness**
  $\longrightarrow$ will my program always produce the right result?
- **Assertions by the programmer**. `assert x > 0;`
  $\longrightarrow$ will this always be true?
- **Memory Safety**
  $\longrightarrow$ are my memory accesses always legal?
  ```
  int* x = NULL;  *x = 42;
  ```
  $\longrightarrow$ undefined behaviour!
  $\longrightarrow$ memory safety matters: Heartbleed (OpenSSL attack)
- . . .

Program verification: the user's perspective (1/3)

What properties of programs do we want to analyse?

- **Partial Correctness**

  $\longrightarrow$ will my program always produce the right result?

- **Assertions by the programmer**. `assert x > 0;`

  $\longrightarrow$ will this always be true?

- **Memory Safety**

  $\longrightarrow$ are my memory accesses always legal?

  ```
  int* x = NULL;  *x = 42;
  ```

  $\longrightarrow$ undefined behaviour!

  $\longrightarrow$ memory safety matters: Heartbleed (OpenSSL attack)

- ...

$\longrightarrow$ Safety properties, at SV-COMP since 2012, for C and Java programs

Program verification: the user's perspective (2/3)

- **Equivalence**. Do **two** programs always produce the same result?
  $\longrightarrow$ correctness of refactoring

Program verification: the user's perspective (2/3)

- **Equivalence**. Do **two** programs always produce the same result?
  $\longrightarrow$ correctness of refactoring

$\longrightarrow$ No (?) competition (yet?)

Program verification: the user's perspective (2/3)

- **Equivalence**. Do **two** programs always produce the same result?
  $\longrightarrow$ correctness of refactoring

$\longrightarrow$ No (?) competition (yet?)

- **Confluence**. For languages with non-deterministic rules/commands: does **one** program always produce the same result?

  Confluence is a property that establishes the global determinism of a computation despite possible local non-determinism.
  [Hristakiev, *PhD thesis '17*]

  $\longrightarrow$ does the order of applying compiler optimisation rules matter?

Program verification: the user's perspective (2/3)

- **Equivalence**. Do **two** programs always produce the same result?
  $\longrightarrow$ correctness of refactoring

$\longrightarrow$ No (?) competition (yet?)

- **Confluence**. For languages with non-deterministic rules/commands: does **one** program always produce the same result?

  > *Confluence is a property that establishes the global determinism of a computation despite possible local non-determinism.*
  > [Hristakiev, *PhD thesis '17*]

  $\longrightarrow$ does the order of applying compiler optimisation rules matter?

$\longrightarrow$ CoCo, since 2012, for term rewrite systems

Program verification: the user's perspective (3/3)

- **Termination**
  $\longrightarrow$ will my program give an output for all inputs
  in finitely many steps?

Program verification: the user's perspective (3/3)

- **Termination**

  $\longrightarrow$ will my program give an output for all inputs in finitely many steps?

$\longrightarrow$ termCOMP, since 2004, for term rewrite systems; integer transition systems; Prolog, Haskell, Java, C programs

Program verification: the user's perspective (3/3)

- **Termination**
  $\longrightarrow$ will my program give an output for all inputs in finitely many steps?

$\longrightarrow$ termCOMP, since 2004, for term rewrite systems; integer transition systems; Prolog, Haskell, Java, C programs

$\longrightarrow$ SV-COMP, since 2014, for C programs

Program verification: the user's perspective (3/3)

- **Termination**
  $\longrightarrow$ will my program give an output for all inputs in finitely many steps?

$\longrightarrow$ termCOMP, since 2004, for term rewrite systems; integer transition systems; Prolog, Haskell, Java, C programs

$\longrightarrow$ SV-COMP, since 2014, for C programs

- **(Quantitative) Resource Use** aka **Complexity**
  $\longrightarrow$ how many steps will my program need in the worst case? (runtime complexity)
  $\longrightarrow$ how large can my data become? (size complexity)

Program verification: the user's perspective (3/3)

- **Termination**

  $\longrightarrow$ will my program give an output for all inputs in finitely many steps?

$\longrightarrow$ termCOMP, since 2004, for term rewrite systems; integer transition systems; Prolog, Haskell, Java, C programs

$\longrightarrow$ SV-COMP, since 2014, for C programs

- **(Quantitative) Resource Use** aka **Complexity**

  $\longrightarrow$ how many steps will my program need in the worst case? (runtime complexity)

  $\longrightarrow$ how large can my data become? (size complexity)

$\longrightarrow$ termCOMP, since 2008, for term rewrite systems; integer transition systems; C programs

Program verification: the user's perspective (3/3)

- **Termination**

  $\longrightarrow$ will my program give an output for all inputs in finitely many steps?

$\longrightarrow$ termCOMP, since 2004, for term rewrite systems; integer transition systems; Prolog, Haskell, Java, C programs

$\longrightarrow$ SV-COMP, since 2014, for C programs

- **(Quantitative) Resource Use** aka **Complexity**

  $\longrightarrow$ how many steps will my program need in the worst case? (runtime complexity)

  $\longrightarrow$ how large can my data become? (size complexity)

$\longrightarrow$ termCOMP, since 2008, for term rewrite systems; integer transition systems; C programs

Note: All these properties are **undecidable**!

Program verification: the user's perspective (3/3)

- **Termination**

  $\longrightarrow$ will my program give an output for all inputs in finitely many steps?

$\longrightarrow$ termCOMP, since 2004, for term rewrite systems; integer transition systems; Prolog, Haskell, Java, C programs

$\longrightarrow$ SV-COMP, since 2014, for C programs

- **(Quantitative) Resource Use** aka **Complexity**

  $\longrightarrow$ how many steps will my program need in the worst case? (runtime complexity)

  $\longrightarrow$ how large can my data become? (size complexity)

$\longrightarrow$ termCOMP, since 2008, for term rewrite systems; integer transition systems; C programs

Note: All these properties are **undecidable**!

$\Rightarrow$ tools use automatable **sufficient criteria**

Input languages: backend languages

Simple, desugared formalisms with clear semantics

Input languages: backend languages

Simple, desugared formalisms with clear semantics

- Term rewrite systems (many flavours)

Input languages: backend languages

Simple, desugared formalisms with clear semantics

- Term rewrite systems (many flavours)
- Integer transition systems

Input languages: backend languages

Simple, desugared formalisms with clear semantics

- Term rewrite systems (many flavours)
- Integer transition systems

Benefits and drawbacks:

Input languages: backend languages

Simple, desugared formalisms with clear semantics

- Term rewrite systems (many flavours)
- Integer transition systems

Benefits and drawbacks:

- Easy(-ish) to agree on semantics

Input languages: backend languages

Simple, desugared formalisms with clear semantics

- Term rewrite systems (many flavours)
- Integer transition systems

Benefits and drawbacks:

- Easy(-ish) to agree on semantics
- Interesting mainly for researchers (rather than programmers)

Input languages: backend languages

Simple, desugared formalisms with clear semantics

- Term rewrite systems (many flavours)
- Integer transition systems

Benefits and drawbacks:

- Easy(-ish) to agree on semantics
- Interesting mainly for researchers (rather than programmers)
- Used internally in verification tools for real-world languages

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, ...

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, . . .

Benefits:

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, . . .

Benefits:

- Lots of real-world examples to analyse

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, ...

Benefits:

- Lots of real-world examples to analyse
- Tools for these languages useful for non-experts (programmers!)

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, . . .

Benefits:

- Lots of real-world examples to analyse
- Tools for these languages useful for non-experts (programmers!)

Challenges:

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, . . .

Benefits:

- Lots of real-world examples to analyse
- Tools for these languages useful for non-experts (programmers!)

Challenges:

- Language semantics can be ambiguous; undefined behaviour ($\longrightarrow$ C)

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, ...

Benefits:

- Lots of real-world examples to analyse
- Tools for these languages useful for non-experts (programmers!)

Challenges:

- Language semantics can be ambiguous; undefined behaviour ($\longrightarrow$ C)
- Tools usually support only language subsets
  (libraries, concurrency, reflection, ...)

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, ...

Benefits:

- Lots of real-world examples to analyse
- Tools for these languages useful for non-experts (programmers!)

Challenges:

- Language semantics can be ambiguous; undefined behaviour ($\longrightarrow$ C)
- Tools usually support only language subsets (libraries, concurrency, reflection, ...)
- Tools may make simplifying assumptions (e.g., treat `int` as $\mathbb{Z}$?)

Input languages: frontend languages

Programming languages used outside of research:

Prolog, Haskell, Java, C, ...

Benefits:

- Lots of real-world examples to analyse
- Tools for these languages useful for non-experts (programmers!)

Challenges:

- Language semantics can be ambiguous; undefined behaviour ($\longrightarrow$ C)
- Tools usually support only language subsets
  (libraries, concurrency, reflection, ...)
- Tools may make simplifying assumptions (e.g., treat `int` as $\mathbb{Z}$?)
- Different competitions may make different assumptions
  (...which make sense in context)

Output format

CoCo, SV-COMP, termCOMP:
Encourage/require machine-checkable verification **certificates** as outputs

Output format

CoCo, SV-COMP, termCOMP:
Encourage/require machine-checkable verification **certificates** as outputs

Correctness checks:

- termCOMP/CoCo:
  checkers based on formalisations of problems + verification techniques in Coq/Isabelle/...

Output format

CoCo, SV-COMP, termCOMP:
Encourage/require machine-checkable verification **certificates** as outputs

Correctness checks:

- termCOMP/CoCo:
  checkers based on formalisations of problems + verification techniques in Coq/Isabelle/...
  $\longrightarrow$ trustable verification tool, no proof search

Output format

CoCo, SV-COMP, termCOMP:
Encourage/require machine-checkable verification **certificates** as outputs

Correctness checks:

- termCOMP/CoCo:
  checkers based on formalisations of problems + verification techniques in Coq/Isabelle/. . .
  $\longrightarrow$ trustable verification tool, no proof search
  $\longrightarrow$ currently for **backend languages** only

Output format

CoCo, SV-COMP, termCOMP:
Encourage/require machine-checkable verification **certificates** as outputs

Correctness checks:

- termCOMP/CoCo:
  checkers based on formalisations of problems + verification techniques in Coq/Isabelle/...
  $\longrightarrow$ trustable verification tool, no proof search
  $\longrightarrow$ currently for **backend languages** only
  $\longrightarrow$ CeTA, talk René Thiemann at WG3 meeting in Valencia, Feb 2022

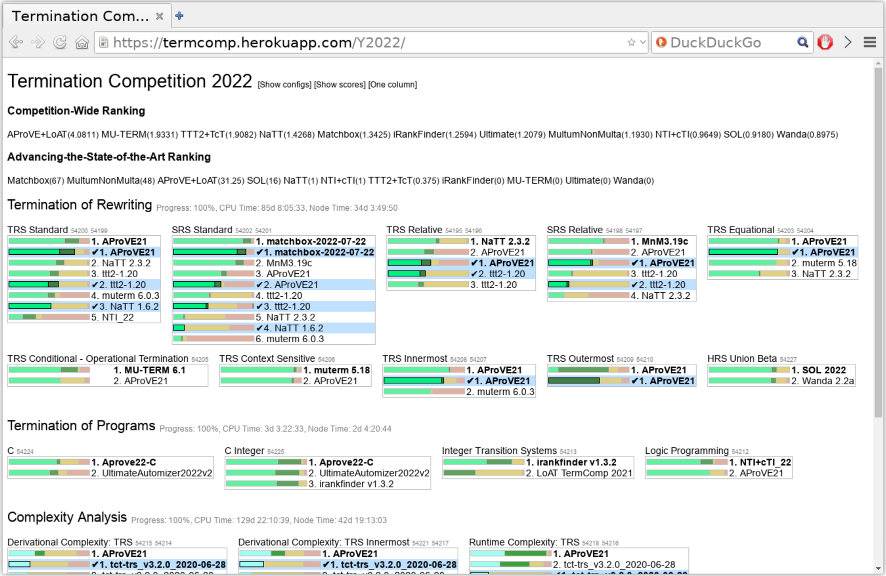
Output format

CoCo, SV-COMP, termCOMP:
Encourage/require machine-checkable verification **certificates** as outputs

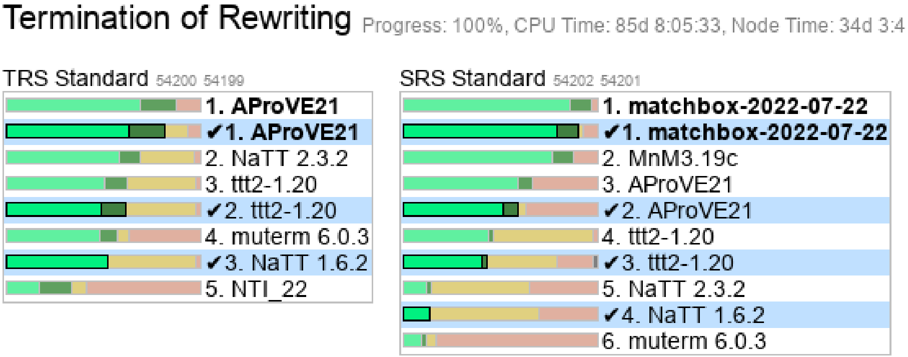
Correctness checks:

- termCOMP/CoCo:
  checkers based on formalisations of problems + verification techniques in Coq/Isabelle/...
  $\longrightarrow$ trustable verification tool, no proof search
  $\longrightarrow$ currently for **backend languages** only
  $\longrightarrow$ CeTA, talk René Thiemann at WG3 meeting in Valencia, Feb 2022
- SV-COMP: other participating tools, for **frontend languages**

termCOMP with certification (✓) (1/2)

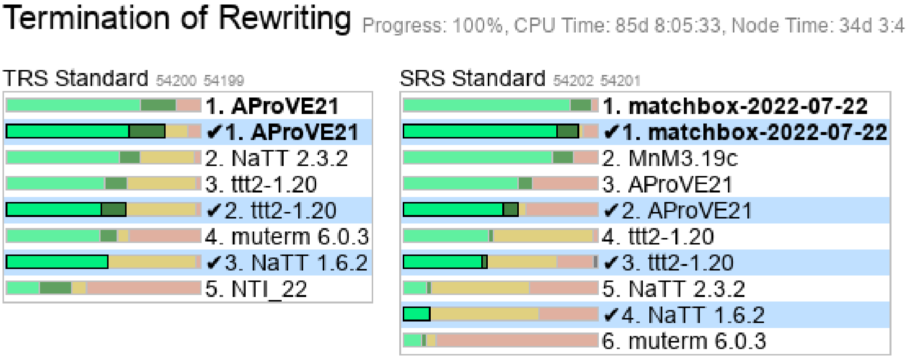


Let's zoom in . . .


Termination of Rewriting
Progress: 100%, CPU Time: 85d 8:05:33, Node Time: 34d 3:4
TRS Standard 54200 54199
1. AProVE21
✔1. AProVE21
2. NaTT 2.3.2
3. ttt2-1.20
✔2. ttt2-1.20
4. muterm 6.0.3
✔3. NaTT 1.6.2
5. NTI_22
SRS Standard 54202 54201
1. matchbox-2022-07-22
✔1. matchbox-2022-07-22
2. MnM3.19c
3. AProVE21
✔2. AProVE21
4. ttt2-1.20
✔3. ttt2-1.20
5. NaTT 2.3.2
✔4. NaTT 1.6.2
6. muterm 6.0.3

termCOMP with certification (✓) (2/2)

Let's zoom in . . .



⇒ proof certification is competitive!

Termination analysis, classically

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")
2. Prove f to have a **lower bound** ("vanish when the machine stops")

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")
2. Prove f to have a **lower bound** ("vanish when the machine stops")
3. Prove that f **decreases** over time

Termination analysis, classically

Turing 1949

Finally the checker has to verify that the process comes to an end. Here again he should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops.

"Finally the checker has to verify that the process comes to an end. [...] This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

1. Find **ranking function** f ("quantity")
2. Prove f to have a **lower bound** ("vanish when the machine stops")
3. Prove that f **decreases** over time

Example (Termination can be simple)

while $x > 0$:
$x = x - 1$

Termination analysis, with automated reasoning

Question: Does program P terminate?

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Answer:

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Answer:

1. φ satisfiable, model M (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow P$ terminating, M fills in the gaps in the termination proof

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Answer:

1. φ satisfiable, model M (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow$ P terminating, M fills in the gaps in the termination proof
2. φ unsatisfiable:
   $\Rightarrow$ termination status of P unknown
   $\Rightarrow$ try a different template (proof technique)

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Answer:

1. φ satisfiable, model M (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow$ P terminating, M fills in the gaps in the termination proof
2. φ unsatisfiable:
   $\Rightarrow$ termination status of P unknown
   $\Rightarrow$ try a different template (proof technique)

In practice:

- Encode only one proof **step** at a time
  $\longrightarrow$ prove only **part** of the program terminating, get simpler problem

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Answer:

1. φ satisfiable, model M (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow$ P terminating, M fills in the gaps in the termination proof
2. φ unsatisfiable:
   $\Rightarrow$ termination status of P unknown
   $\Rightarrow$ try a different template (proof technique)

In practice:

- Encode only one proof **step** at a time
  $\longrightarrow$ prove only **part** of the program terminating, get simpler problem
  $\longrightarrow$ combine techniques

Termination analysis, with automated reasoning

Question: Does program P terminate?

Approach: Encode termination proof **template** to logical constraint φ, ask SMT solver

$\longrightarrow$ **SMT** = **SAT**isfiability **M**odulo **T**heories, solve constraints like

$$\varphi \quad = \quad b > 0 \quad \wedge \quad (4\,a\,b - 7\,b^2 > 1 \quad \vee \quad 3\,a + c \geq b^3)$$

Answer:

1. φ satisfiable, model M (e.g., $a = 3, b = 1, c = 1$):
   $\Rightarrow$ P terminating, M fills in the gaps in the termination proof
2. φ unsatisfiable:
   $\Rightarrow$ termination status of P unknown
   $\Rightarrow$ try a different template (proof technique)

In practice:

- Encode only one proof **step** at a time
  $\longrightarrow$ prove only **part** of the program terminating, get simpler problem
  $\longrightarrow$ combine techniques
- **Repeat** until the whole program is proved terminating

Papers on termination of imperative programs often about **integers** as data

Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

if $(\mathsf{x} \geq 0)$
  while $(\mathsf{x} \neq 0)$
    $\mathsf{x} = \mathsf{x} - 1;$

Does this program terminate?
(x ranges over $\mathbb{Z}$)

Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

ℓ_0: **if** $(x \geq 0)$
ℓ_1: **while** $(x \neq 0)$
ℓ_2: $x = x - 1;$

Does this program terminate?
(x ranges over $\mathbb{Z}$)

Example (Equivalent Translation to an Integer Transition System, cf. [McCarthy, *CACM '60*])

$$\begin{array}{lcll}
\ell_0(x) & \rightarrow & \ell_1(x) & [x \geq 0] \\
\ell_0(x) & \rightarrow & \ell_3(x) & [x < 0] \\
\ell_1(x) & \rightarrow & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \rightarrow & \ell_1(x-1) & \\
\ell_1(x) & \rightarrow & \ell_3(x) & [x = 0]
\end{array}$$

Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

ℓ_0: **if** $(\mathsf{x} \geq 0)$
ℓ_1: **while** $(\mathsf{x} \neq 0)$
ℓ_2: $\mathsf{x} = \mathsf{x} - 1;$

Does this program terminate?
(x ranges over $\mathbb{Z}$)

Example (Equivalent Translation to an Integer Transition System, cf. [McCarthy, *CACM '60*])

$$
\begin{array}{lcll}
\ell_0(x) & \rightarrow & \ell_1(x) & [x \geq 0] \\
\ell_0(x) & \rightarrow & \ell_3(x) & [x < 0] \\
\ell_1(x) & \rightarrow & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \rightarrow & \ell_1(x-1) & \\
\ell_1(x) & \rightarrow & \ell_3(x) & [x = 0]
\end{array}
$$

And this one?

Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

ℓ_0: **if** (x ≥ 0)
ℓ_1: **while** (x $\neq 0$)
ℓ_2: x = x − 1;

Does this program terminate?
(x ranges over $\mathbb{Z}$)

Example (Equivalent Translation to an Integer Transition System, cf. [McCarthy, *CACM '60*])

$$\begin{array}{lcll} \ell_0(x) & \rightarrow & \ell_1(x) & [x \geq 0] \\ \ell_0(x) & \rightarrow & \ell_3(x) & [x < 0] \\ \ell_1(x) & \rightarrow & \ell_2(x) & [x \neq 0] \\ \ell_2(x) & \rightarrow & \ell_1(x-1) & \\ \ell_1(x) & \rightarrow & \ell_3(x) & [x = 0] \end{array}$$

And this one?

Oh no! $\ell_1(-1) \rightarrow \ell_2(-1) \rightarrow \ell_1(-2) \rightarrow \ell_2(-2) \rightarrow \ell_1(-3) \rightarrow \cdots$

Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

ℓ_0: **if** $(x \geq 0)$
ℓ_1: **while** $(x \neq 0)$
ℓ_2: $x = x - 1;$

Does this program terminate?
(x ranges over $\mathbb{Z}$)

Example (Equivalent Translation to an Integer Transition System, cf. [McCarthy, *CACM '60*])

$$
\begin{array}{llll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_0(x) & \to & \ell_3(x) & [x < 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & \\
\ell_1(x) & \to & \ell_3(x) & [x = 0]
\end{array}
$$

And this one?

Oh no! $\ell_1(-1) \to \ell_2(-1) \to \ell_1(-2) \to \ell_2(-2) \to \ell_1(-3) \to \cdots$

$\Rightarrow$ **Restrict initial states to** $\ell_0(z)$ for $z \in \mathbb{Z}$

Papers on termination of imperative programs often about **integers** as data

Example (Imperative Program)

ℓ_0: **if** (x ≥ 0)
ℓ_1: **while** (x $\neq 0$)
ℓ_2: x $=$ x $- 1$;

Does this program terminate?
(x ranges over $\mathbb{Z}$)

Example (Equivalent Translation to an Integer Transition System, cf. [McCarthy, *CACM '60*])

$$\begin{array}{lcll} \ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\ \ell_0(x) & \to & \ell_3(x) & [x < 0] \\ \ell_1(x) & \to & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\ \ell_2(x) & \to & \ell_1(x-1) & [x \geq 0] \\ \ell_1(x) & \to & \ell_3(x) & [x = 0 \wedge x \geq 0] \end{array}$$

And this one?

Oh no! $\ell_1(-1) \to \ell_2(-1) \to \ell_1(-2) \to \ell_2(-2) \to \ell_1(-3) \to \cdots$

$\Rightarrow$ **Restrict initial states** to $\ell_0(z)$ for $z \in \mathbb{Z}$

$\Rightarrow$ Find **invariant** $x \geq 0$ at ℓ_1, ℓ_2 (how?)

Proving termination with invariants

Example (Transition system with invariants)

$$
\begin{array}{lcll}
\ell_0(x) & \to & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \to & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \to & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \to & \ell_3(x) & [x = 0 \wedge x \geq 0]
\end{array}
$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{lcll} \ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\ \ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_3(x) & [x = 0 \wedge x \geq 0] \end{array}$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{lcll} \ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\ \ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_3(x) & [x = 0 \wedge x \geq 0] \end{array}$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Proving termination with invariants

Example (Transition system with invariants)

$$
\begin{array}{llll}
\ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \land x \geq 0] \\
\ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_3(x) & [x = 0 \land x \geq 0]
\end{array}
$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints here:

$$
\begin{array}{lll}
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x-1) \\
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0
\end{array}
$$

"decrease ..."

"... against a bound"

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{llll}
\ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\
\ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\
\ell_1(x) & \succsim & \ell_3(x) & [x = 0 \wedge x \geq 0]
\end{array}$$

Prove termination by ranking function $[\cdot]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints here:

$$\begin{array}{lll}
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x-1) \\
x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0
\end{array}$$

"decrease …" / "… against a bound"

Use Farkas' Lemma to eliminate $\forall x$, solver for **linear** constraints gives model for a_i, b_i.

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{llll} \ell_0(x) & \succsim & \ell_1(x) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_2(x) & [x \neq 0 \land x \geq 0] \\ \ell_2(x) & \succ & \ell_1(x-1) & [x \geq 0] \\ \ell_1(x) & \succsim & \ell_3(x) & [x = 0 \land x \geq 0] \end{array}$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints here:

$$\begin{array}{lll} x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x-1) \\ x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0 \end{array}$$

"decrease ..." / "... against a bound"

Use Farkas' Lemma to eliminate $\forall x$, solver for **linear** constraints gives model for a_i, b_i.

More: [Podelski, Rybalchenko, *VMCAI '04*, Alias et al, *SAS '10*]

Proving termination with invariants

Example (Transition system with invariants)

$$\begin{array}{lcll} \ell_0(x) & \rightarrow & \ell_1(x) & [x \geq 0] \\ \ell_1(x) & \rightarrow & \ell_2(x) & [x \neq 0 \wedge x \geq 0] \\ & & & \\ \ell_1(x) & \rightarrow & \ell_3(x) & [x = 0 \wedge x \geq 0] \end{array}$$

Prove termination by ranking function $[\,\cdot\,]$ with $[\ell_0](x) = [\ell_1](x) = \cdots = x$

Automate search using parametric ranking function:

$$[\ell_0](x) = a_0 + b_0 \cdot x, \quad [\ell_1](x) = a_1 + b_1 \cdot x, \quad \ldots$$

Constraints here:

$$\begin{array}{lcll} x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x > a_1 + b_1 \cdot (x - 1) & \text{“decrease \ldots”} \\ x \geq 0 & \Rightarrow & a_2 + b_2 \cdot x \geq 0 & \text{“\ldots against a bound”} \end{array}$$

Use Farkas' Lemma to eliminate $\forall x$, solver for **linear** constraints gives model for a_i, b_i.

More: [Podelski, Rybalchenko, *VMCAI '04*, Alias et al, *SAS '10*]

Searching for invariants using SMT

Termination prover for programs needs invariants ($\longrightarrow$ safety!)

Searching for invariants using SMT

Termination prover for programs needs invariants ($\longrightarrow$ safety!)

- Statically before the translation
  [Otto et al, *RTA '10*; Ströder et al, *JAR '17*, ...]
  $\longrightarrow$ abstract interpretation [Cousot, Cousot, *POPL '77*]

Searching for invariants using SMT

Termination prover for programs needs invariants ($\longrightarrow$ safety!)

- Statically before the translation
  [Otto et al, *RTA '10*; Ströder et al, *JAR '17*, ...]
  $\longrightarrow$ abstract interpretation [Cousot, Cousot, *POPL '77*]

- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\longrightarrow$ prove termination of single program **runs**
  $\longrightarrow$ termination argument often generalises

Searching for invariants using SMT

Termination prover for programs needs invariants ($\longrightarrow$ safety!)

- Statically before the translation
  [Otto et al, *RTA '10*; Ströder et al, *JAR '17*, ...]
  $\longrightarrow$ abstract interpretation [Cousot, Cousot, *POPL '77*]

- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\longrightarrow$ prove termination of single program **runs**
  $\longrightarrow$ termination argument often generalises

- ... also cooperating with removal of terminating rules: **T2**
  [Brockschmidt, Cook, Fuhs, *CAV '13*; Brockschmidt et al, *TACAS '16*]

Searching for invariants using SMT

Termination prover for programs needs invariants ($\longrightarrow$ safety!)

- Statically before the translation
  [Otto et al, *RTA '10*; Ströder et al, *JAR '17*, . . .]
  $\longrightarrow$ abstract interpretation [Cousot, Cousot, *POPL '77*]

- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\longrightarrow$ prove termination of single program **runs**
  $\longrightarrow$ termination argument often generalises

- . . . also cooperating with removal of terminating rules: **T2**
  [Brockschmidt, Cook, Fuhs, *CAV '13*; Brockschmidt et al, *TACAS '16*]

- Using Max-SMT
  [Larraz, Oliveras, Rodríguez-Carbonell, Rubio, *FMCAD '13*]

Searching for invariants using SMT

Termination prover for programs needs invariants ($\longrightarrow$ safety!)

- Statically before the translation
  [Otto et al, *RTA '10*; Ströder et al, *JAR '17*, . . .]
  $\longrightarrow$ abstract interpretation [Cousot, Cousot, *POPL '77*]

- By counterexample-based reasoning + safety prover: **Terminator**
  [Cook, Podelski, Rybalchenko, *CAV '06, PLDI '06*]
  $\longrightarrow$ prove termination of single program **runs**
  $\longrightarrow$ termination argument often generalises

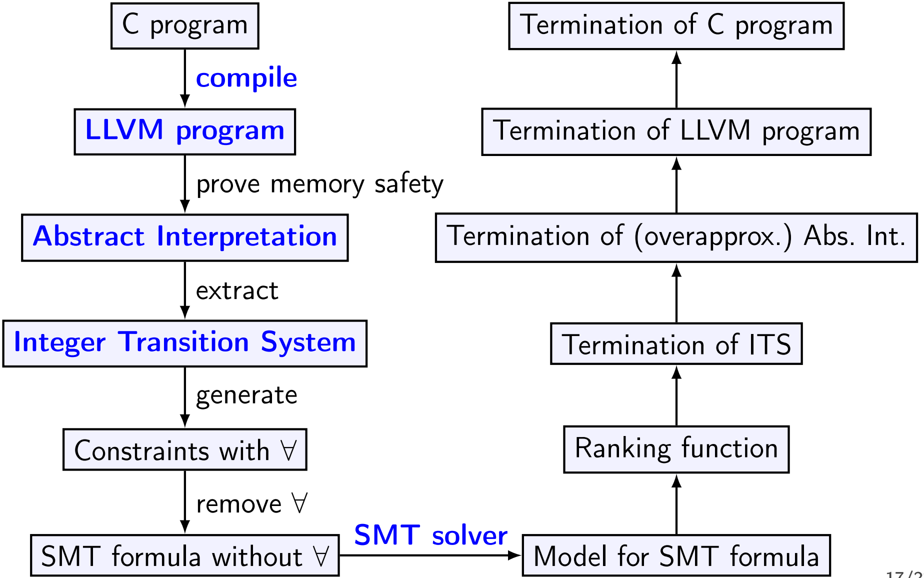
- . . . also cooperating with removal of terminating rules: **T2**
  [Brockschmidt, Cook, Fuhs, *CAV '13*; Brockschmidt et al, *TACAS '16*]

- Using Max-SMT
  [Larraz, Oliveras, Rodríguez-Carbonell, Rubio, *FMCAD '13*]

Nowadays all SMT-based!

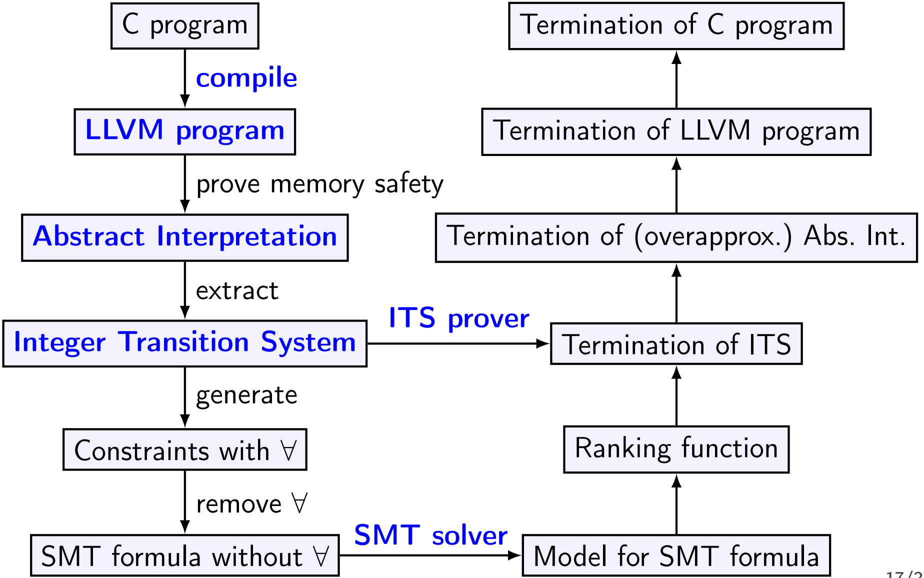
WG3: what is interesting for our tool inventory?

Example: AProVE's termination analysis for C **in detail**



WG3: what is interesting for our tool inventory?

Example: AProVE's termination analysis for C **in detail**



Towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?

Towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?
- What properties can be verified?

Towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?
- What properties can be verified?
- What are the tool's main techniques for the supported **(input, property)** pairs?

Towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?
- What properties can be verified?
- What are the tool's main techniques for the supported **(input, property)** pairs?
- What external tools are used? ($\longrightarrow$ compilers, SMT solvers, ...)

Towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?
- What properties can be verified?
- What are the tool's main techniques for the supported **(input, property)** pairs?
- What external tools are used? ($\longrightarrow$ compilers, SMT solvers, ...)
- What is the tool's URL?

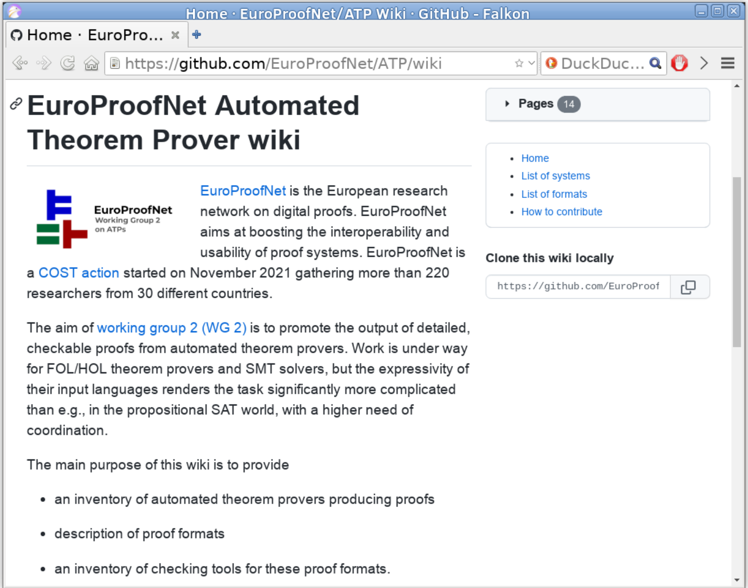
Towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?
- What properties can be verified?
- What are the tool's main techniques for the supported **(input, property)** pairs?
- What external tools are used? ($\longrightarrow$ compilers, SMT solvers, ...)
- What is the tool's URL?
- What is the "canonical reference" to a system description?

Inspiration: WG2's inventory of ATPs (1/3)

Inspiration: https://github.com/EuroProofNet/ATP/wiki


Home · EuroProofNet/ATP Wiki · GitHub - Falkon

Home · EuroPro...

https://github.com/EuroProofNet/ATP/wiki

DuckDuc...

EuroProofNet Automated Theorem Prover wiki

EuroProofNet
Working Group 2
on ATPs

EuroProofNet is the European research network on digital proofs. EuroProofNet aims at boosting the interoperability and usability of proof systems. EuroProofNet is a COST action started on November 2021 gathering more than 220 researchers from 30 different countries.

The aim of working group 2 (WG 2) is to promote the output of detailed, checkable proofs from automated theorem provers. Work is under way for FOL/HOL theorem provers and SMT solvers, but the expressivity of their input languages renders the task significantly more complicated than e.g., in the propositional SAT world, with a higher need of coordination.

The main purpose of this wiki is to provide

- an inventory of automated theorem provers producing proofs
- description of proof formats
- an inventory of checking tools for these proof formats.

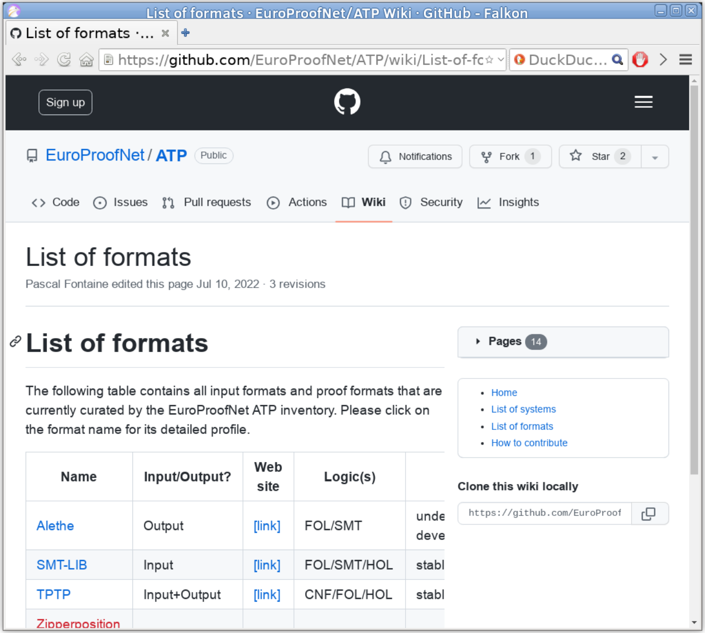
Pages 14

- Home
- List of systems
- List of formats
- How to contribute

Clone this wiki locally

https://github.com/EuroProof

Inspiration: WG2's inventory of ATPs (2/3)



List of formats · EuroProofNet/ATP Wiki · GitHub - Falkon

https://github.com/EuroProofNet/ATP/wiki/List-of-fc

Sign up

EuroProofNet / **ATP** Public

Notifications | Fork 1 | Star 2

Code | Issues | Pull requests | Actions | **Wiki** | Security | Insights

List of formats

Pascal Fontaine edited this page Jul 10, 2022 · 3 revisions

List of formats

The following table contains all input formats and proof formats that are currently curated by the EuroProofNet ATP inventory. Please click on the format name for its detailed profile.

Name	Input/Output?	Web site	Logic(s)	
Alethe	Output	[link]	FOL/SMT	unde deve
SMT-LIB	Input	[link]	FOL/SMT/HOL	stabl
TPTP	Input+Output	[link]	CNF/FOL/HOL	stabl
Zipperposition				

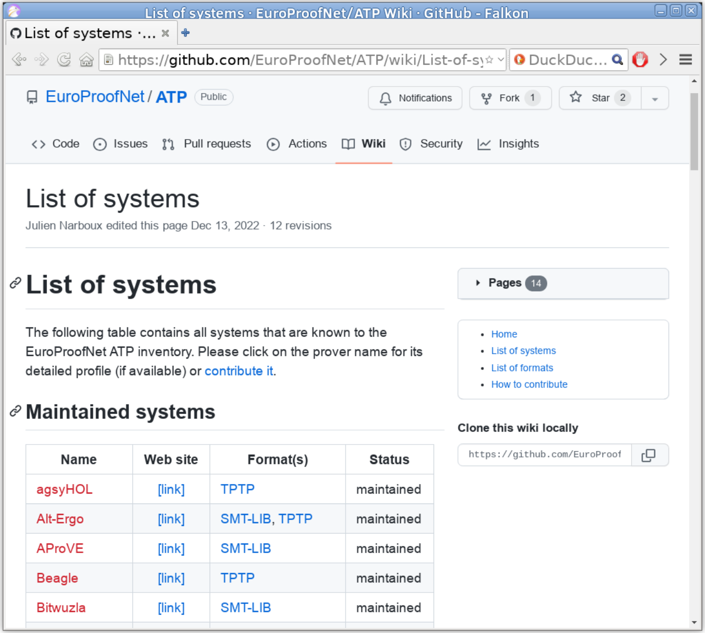
Pages 14

- Home
- List of systems
- List of formats
- How to contribute

Clone this wiki locally

https://github.com/EuroProof

Inspiration: WG2's inventory of ATPs (3/3)


List of systems · EuroProofNet/ATP Wiki · GitHub - Falkon

List of systems ·…

https://github.com/EuroProofNet/ATP/wiki/List-of-sy

DuckDuc…

EuroProofNet / ATP Public

Notifications Fork 1 Star 2

Code Issues Pull requests Actions Wiki Security Insights

List of systems

Julien Narboux edited this page Dec 13, 2022 · 12 revisions

List of systems

The following table contains all systems that are known to the EuroProofNet ATP inventory. Please click on the prover name for its detailed profile (if available) or contribute it.

Maintained systems

Name	Web site	Format(s)	Status
agsyHOL	[link]	TPTP	maintained
Alt-Ergo	[link]	SMT-LIB, TPTP	maintained
AProVE	[link]	SMT-LIB	maintained
Beagle	[link]	TPTP	maintained
Bitwuzla	[link]	SMT-LIB	maintained

Pages 14

- Home
- List of systems
- List of formats
- How to contribute

Clone this wiki locally

https://github.com/EuroProof

Conclusion: towards the first deliverable

Tool inventory of verification tools

- What inputs are supported?
- What properties can be verified?
- What are the tool's main techniques for the supported **(input, property)** pairs?
- What external tools are used? ($\longrightarrow$ compilers, SMT solvers, . . .)
- What is the tool's URL?
- What is the "canonical reference" to a system description?