LTL Software Model Checking in CPAchecker

Thomas Bunk

March 27, 2019

LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

SoSy-Lab
Software Systems

Outline

- Motivation
- Example
- LTL Software Model Checking in CPAchecker
- Trace Abstraction
- Outlook

Motivation

LTL properties already used in SV-Comp, e.g.:

- Unreachability of Error Function:
  `CHECK( init(main()), LTL(G ! call(__VERIFIER_error())) )`
- Memory Safety:
  `CHECK( init(main()), LTL(G valid-free) )`

Motivation

LTL properties already used in SV-Comp, e.g.:

- Unreachability of Error Function:
  ```
  CHECK( init(main()), LTL(G ! call(__VERIFIER_error())) )
  ```
- Memory Safety:
  ```
  CHECK( init(main()), LTL(G valid-free) )
  ```

⇒ Safety properties of the form "*nothing bad will ever happen*"

Motivation

LTL properties already used in SV-Comp, e.g.:

- Unreachability of Error Function:
  `CHECK( init(main()), LTL(G ! call(__VERIFIER_error())) )`
- Memory Safety:
  `CHECK( init(main()), LTL(G valid-free) )`

⇒ Safety properties of the form "*nothing bad will ever happen*"

Formal definition [1]:

$$\forall \sigma \in S^\omega : \ \sigma \vDash \varphi \qquad \text{iff } \forall i \geq 0 : \ \exists w \in S^\omega : \ \sigma[0..i]w \vDash \varphi$$

for safety property φ and a set of program states S.

Liveness properties

They assert that "*something good will happen eventually*"

- Liveness properties already used in SV-Comp, e.g.:
  - Termination: `CHECK( init(main()), LTL(F end) )`

Liveness properties

They assert that "*something good will happen eventually*"

- Liveness properties already used in SV-Comp, e.g.:
    - Termination: `CHECK( init(main()), LTL(F end) )`

- A computation that violates the property can never so after a finite number of transitions
- Any finite prefix can be extended such that the resulting infinite trace satisfies the LT property

Liveness properties

They assert that "*something good will happen eventually*"

- Liveness properties already used in SV-Comp, e.g.:
  - Termination: `CHECK( init(main()), LTL(F end) )`

- A computation that violates the property can never so after a finite number of transitions
- Any finite prefix can be extended such that the resulting infinite trace satisfies the LT property

Formally [1]:

$$\boxed{\forall \sigma \in S^* : \ \sigma \vDash \varphi \qquad \text{iff } \exists \beta \in S^\omega : \ \sigma\beta \vDash \varphi}$$

for liveness property φ and a set of program states S.

Outline

- Motivation
- **Example**
- LTL Software Model Checking in CPAchecker
- Trace Abstraction
- Outlook

Example

```
int x, y;
while (true) {
   x := *;
   y := 1;
   while (x > 0) {
      x--;
      if (x <= 1) {
         y := 0;
      }
   }
}
```

Listing 1: Program P as pseudocode

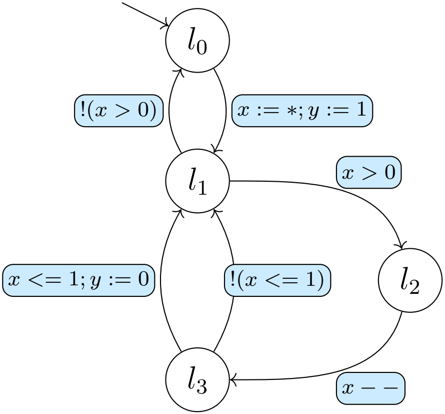


Figure: CFA of program P

Combining the CFA and LT property

```
int x, y;
while (true) {
  x := *;
  y := 1;
  while (x > 0) {
    x--;
    if (x <= 1) {
      y := 0;
    }
  }
}
```

Listing 2: Program P

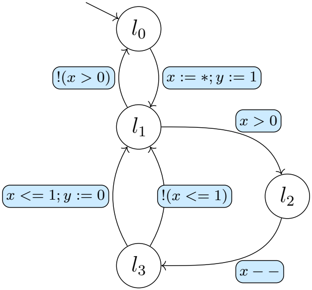


Figure: CFA of P

LTL property:
$\varphi = \Box(x > 0 \rightarrow \Diamond(y = 0))$

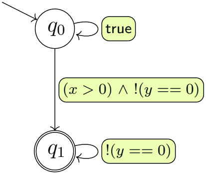
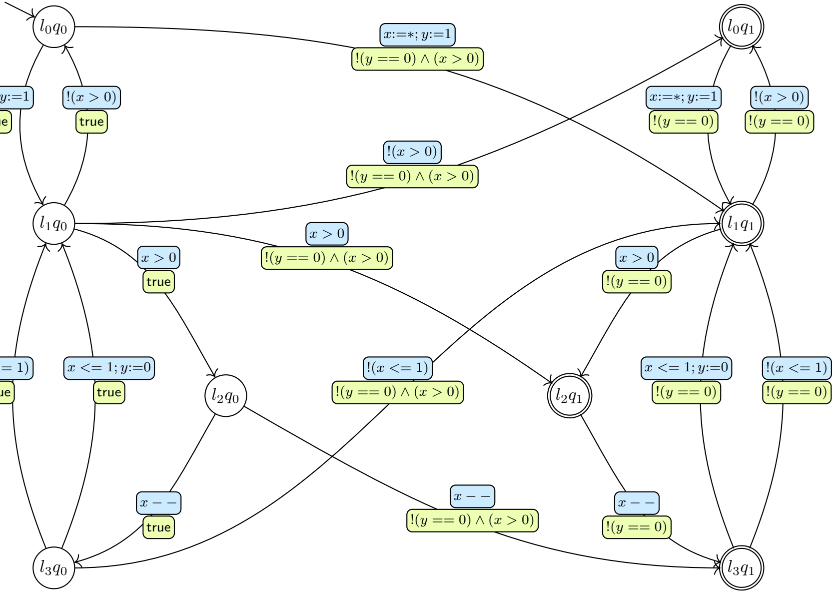
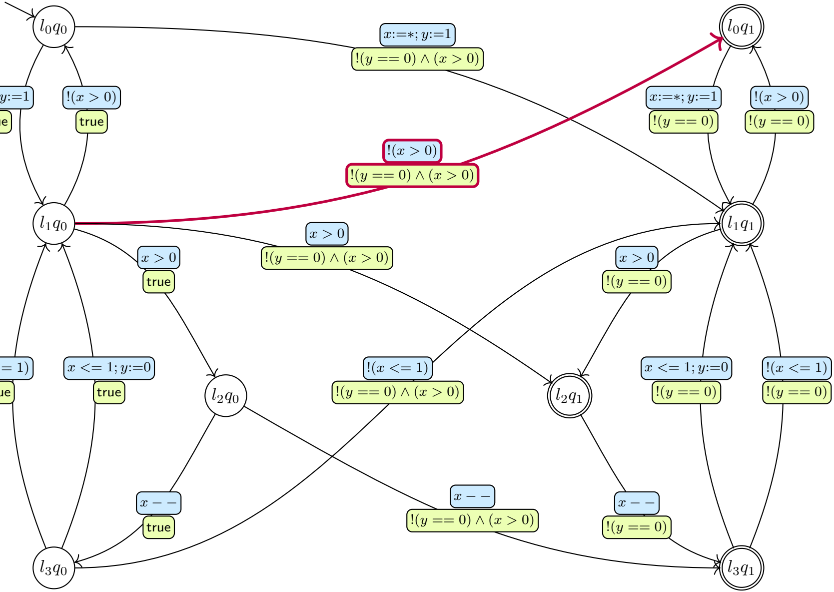


Figure: Büchi Automaton $A_{\neg\varphi}$

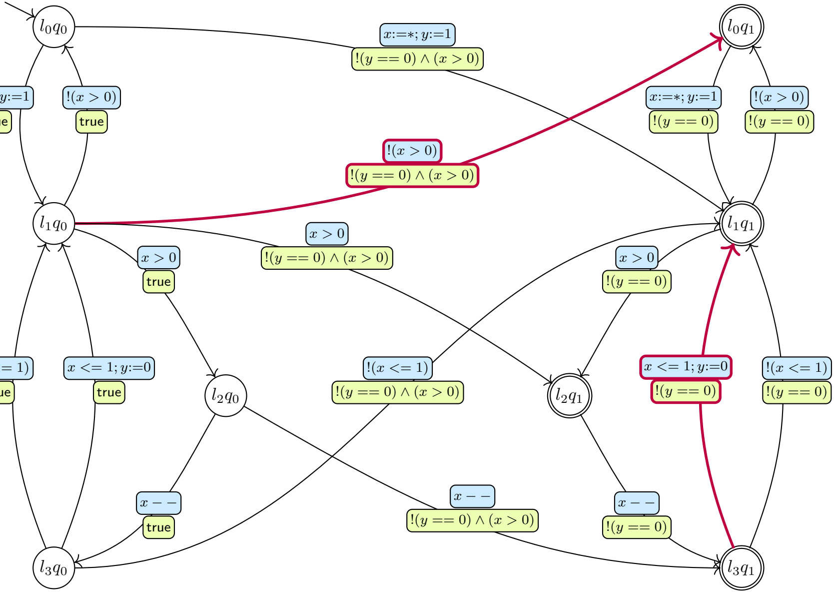
Büchi-program



Local infeasibility



Local infeasibility



Feasibility of statements

In general:

- show that infinite sequence of statements along a path is not executable

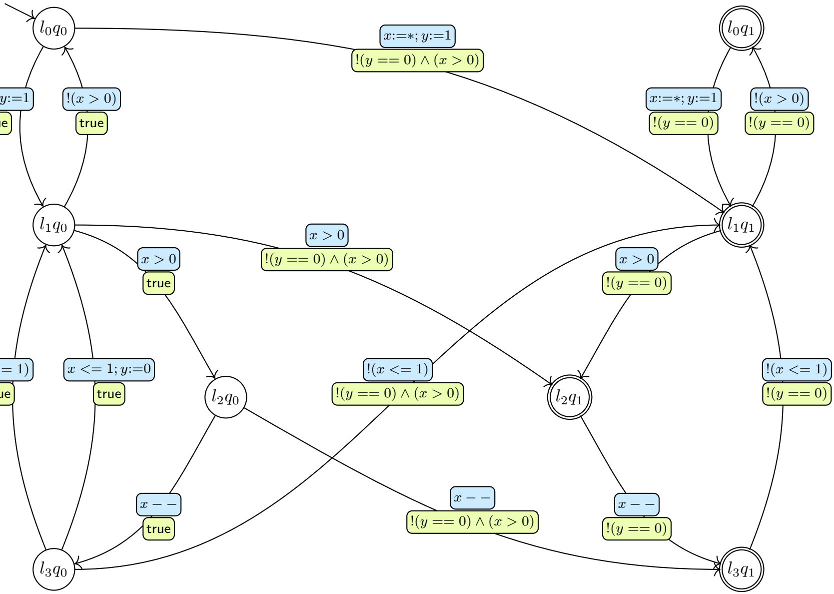
Example:

τ_1: $x--$ x > y $x--$ x > y $x--$ x > y $x--$ $\cdots$

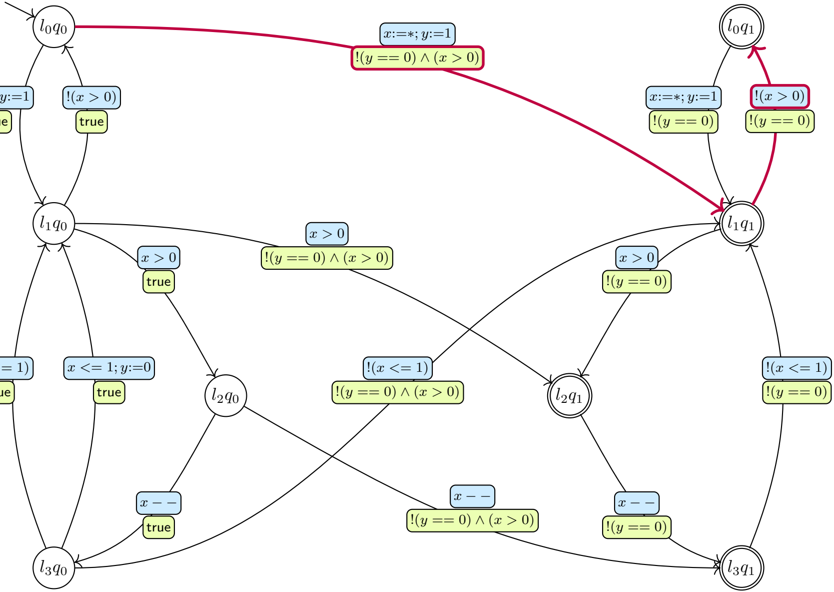
τ_2: $x := y$ x > y $x--$ x > y $x--$ x > y $x--$ $\cdots$

> “Construction of ranking function is always more costly than a proof for unsatisfiability.” [2]

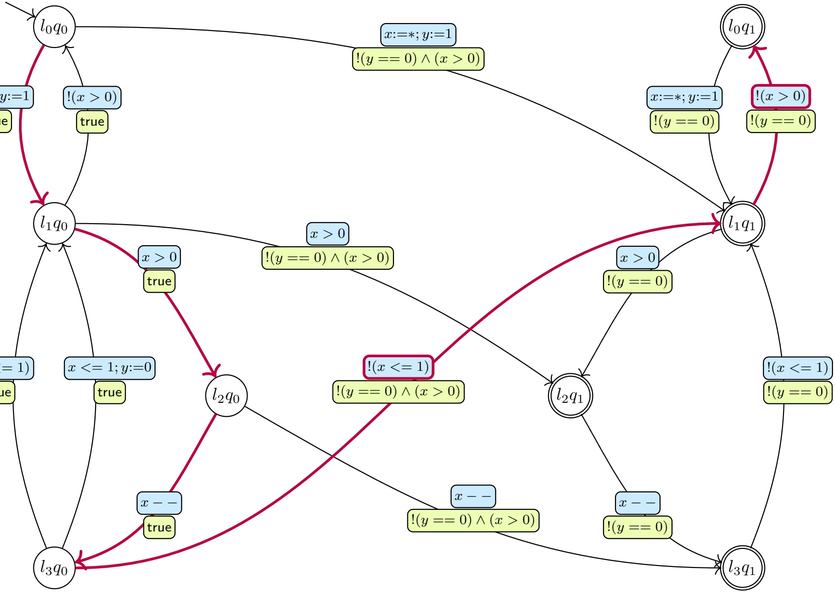
Infeasibility of a finite prefix



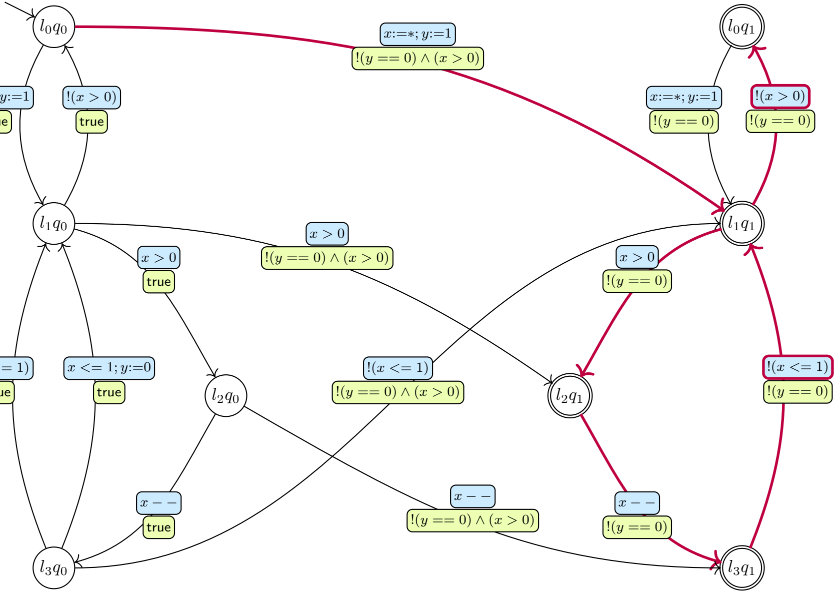
Infeasibility of a finite prefix



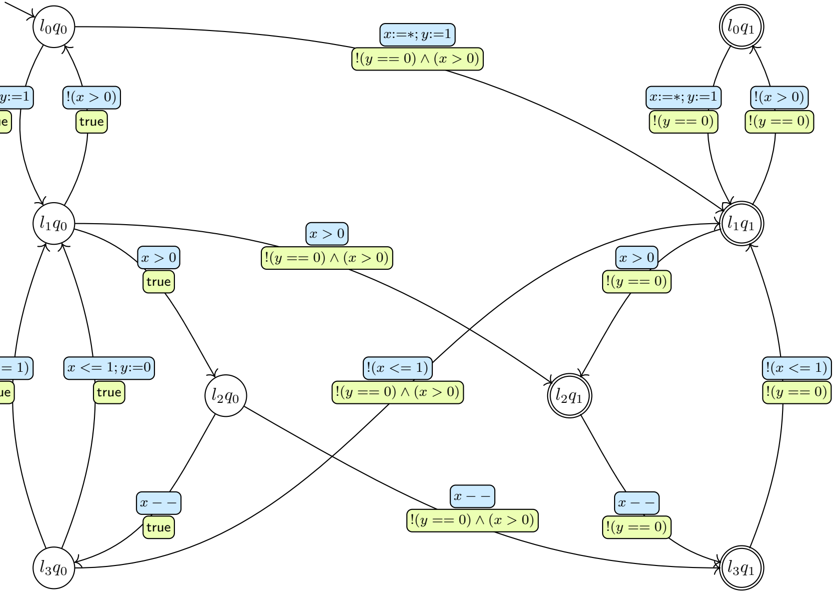
Infeasibility of a finite prefix



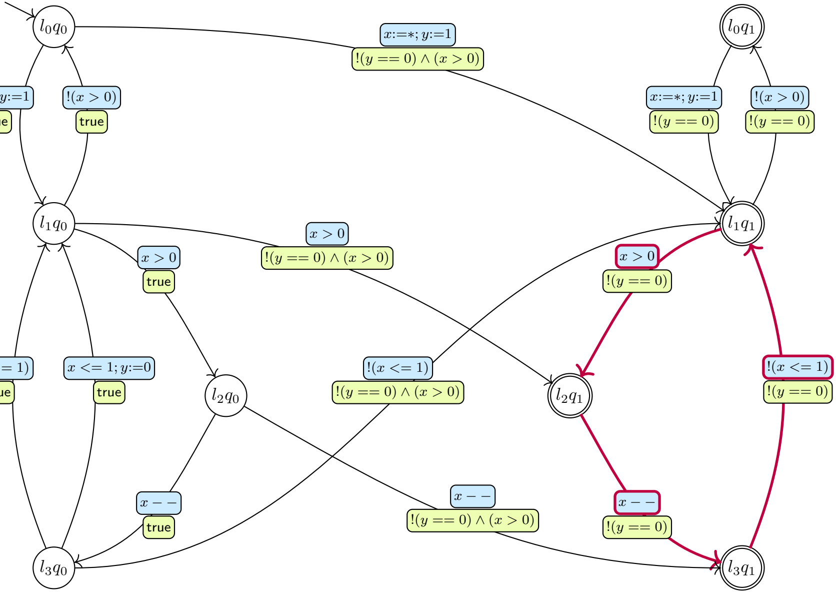
Infeasibility of a finite prefix



ω-Infeasibility



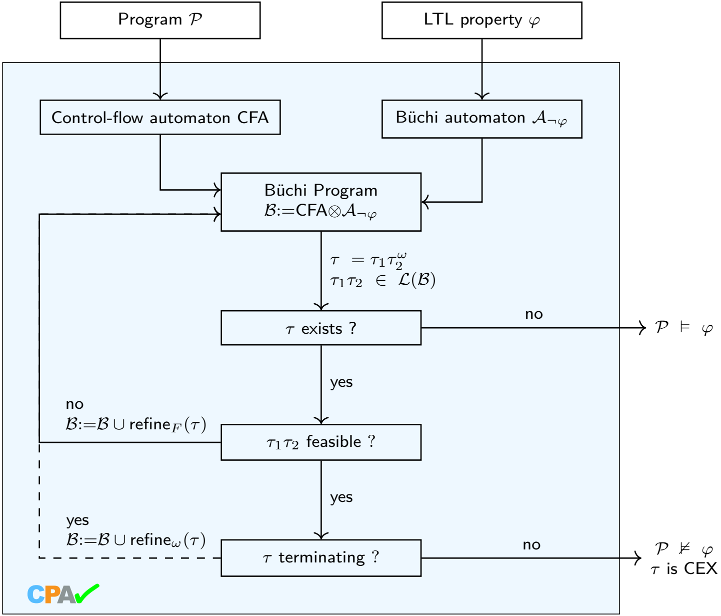
ω-Infeasibility



Outline

- Motivation
- Example
- **LTL Software Model Checking in CPAchecker**
- Trace Abstraction
- Outlook

LTL Software Model Checking



Program $\mathcal{P}$

LTL property φ

Control-flow automaton CFA

Büchi automaton $\mathcal{A}_{\neg\varphi}$

Büchi Program
$\mathcal{B}$:=CFA$\otimes\mathcal{A}_{\neg\varphi}$

$\tau = \tau_1\tau_2^{\omega}$
$\tau_1\tau_2 \in \mathcal{L}(\mathcal{B})$

τ exists ?

no

$\mathcal{P} \models \varphi$

yes

no
$\mathcal{B}$:=$\mathcal{B} \cup$ refine$_F(\tau)$

$\tau_1\tau_2$ feasible ?

yes

yes
$\mathcal{B}$:=$\mathcal{B} \cup$ refine$_\omega(\tau)$

τ terminating ?

no

$\mathcal{P} \not\models \varphi$
τ is CEX

What has been done in CPAchecker (1/3)

- Framework to transform LTL-formulas into automatons from CPAchecker-framework, including:
  - Various source-to-source transformations that optimize and simplify the input property

Transforming LTL-formulas to CPAchecker automatons

- Parsing of LTL properties
  - Consider e.g.: [] (a -> F b U X "c > 0")
- Transforming the result into Büchi automatons
- Parsing and converting the output from the external tool into automatons from CPAchecker-framework

Transforming LTL-formulas to CPAchecker automatons

- Parsing of LTL properties
  - Consider e.g.: [] (a -> F b U X "c > 0")
- Transforming the result into Büchi automatons
- Parsing and converting the output from the external tool into automatons from CPAchecker-framework

- Example formula with 146 characters:

```
G((p0 & Fp1) -> ((!p1 & !p2) U (p1 | ((!p1 & p2 | p1 R p2) U (p1 | ((!p1 & !p2) U ((p3 -> p1) & p1 | ((!p1 & p2) U (p1 | (!p2 U p1))))))))))
```

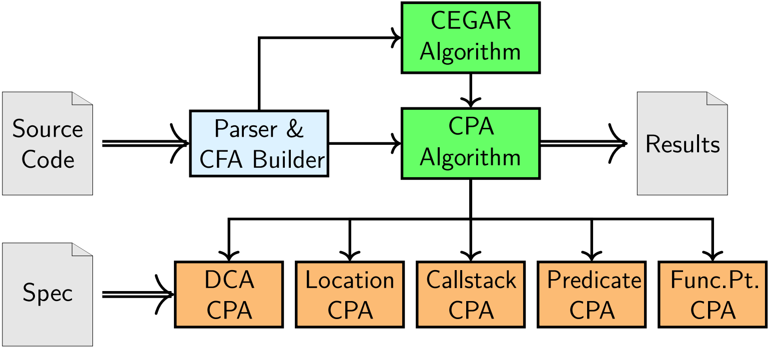
⇒ Total time from raw LTL formula to CPAchecker automaton: 0.194s

What has been done in CPAchecker (2/3)

- Framework to transform LTL-formulas into automatons from CPAchecker-framework, including:
    - Various source-to-source transformations that optimize and simplify the input property

- Implementation of an LTL software model checking algorithm using already existing components in CPAchecker, e.g. :
    - Parser for ANSI C
    - CPA-framework
    - Ranking function synthesis algorithm
    - ...

CPAchecker: Architecture



What has been done in CPAchecker (3/3)

- Framework to transform LTL-formulas into automatons from CPAchecker-framework, including:
    - Various source-to-source transformations that optimize and simplify the input property
- Implementation of an LTL software model checking algorithm using already existing components in CPAchecker, e.g. :
    - Parser for ANSI C
    - CPA-framework
    - Ranking function synthesis algorithm
    - ...
- Implementation of a *Trace Abstraction algorithm*

Outline

- Motivation
- Example
- LTL Software Model Checking in CPAchecker
- **Trace Abstraction**
- Outlook

Trace abstraction

Definition [5]:

A trace abstraction is given by a tuple of automatons $(\mathcal{A}_1, ..., \mathcal{A}_n)$ such that $\mathcal{A}_i$ recognizes a subset of infeasible traces for $i = 1, ..., n$.

We say that the trace abstraction $(\mathcal{A}_1, ..., \mathcal{A}_n)$ does not admit an error trace if $\mathcal{A}_{\mathcal{P}} \cap \overline{\mathcal{A}_1} \cap ... \cap \overline{\mathcal{A}_n}$ is empty.

Example for Trace Abstraction

```
x := 0;
y := 0;
while(nondet) {x++;}
assert(x != -1);
assert(y != -1);
```

Listing 3: Program P as pseudocode

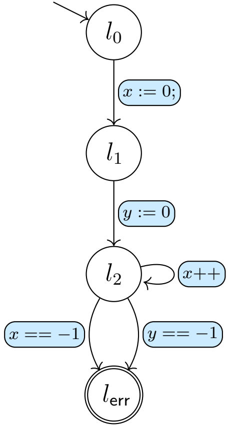


Figure: CFA of program P

Trace Abstraction – Interpolant Based Approach

- Generalize infeasible error traces
- Exclude classes of infeasible traces

Example – Interpolants for trace τ:

$$\boxed{x := 0} \qquad \boxed{y := 0} \qquad \boxed{x{+}{+}} \qquad \boxed{x == -1}$$

Trace Abstraction – Interpolant Based Approach

- Generalize infeasible error traces
- Exclude classes of infeasible traces

Example – Interpolants for trace τ:

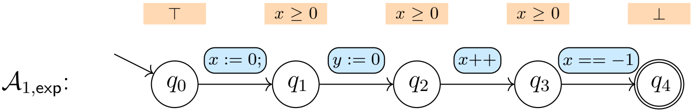
$\top$ $\boxed{x := 0}$ $x \geq 0$ $\boxed{y := 0}$ $x \geq 0$ $\boxed{x{+}{+}}$ $x \geq 0$ $\boxed{x == -1}$ $\bot$

Trace Abstraction – Interpolant Based Approach

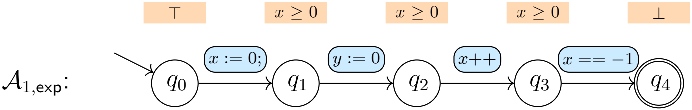
- Generalize infeasible error traces
- Exclude classes of infeasible traces

Example – Interpolants for trace τ:

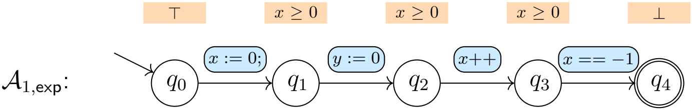
$\top$ $x := 0$ $x \geq 0$ $y := 0$ $x \geq 0$ $x{+}{+}$ $x \geq 0$ $x == -1$ $\bot$

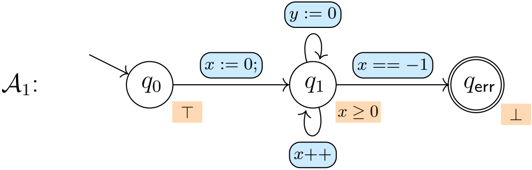


Trace Abstraction – Interpolant Based Approach



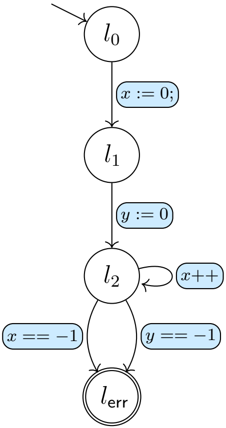
Trace Abstraction – Interpolant Based Approach



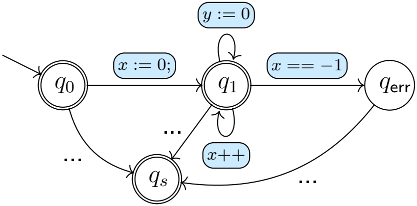


Trace Abstraction – Interpolant Based Approach

CFA:

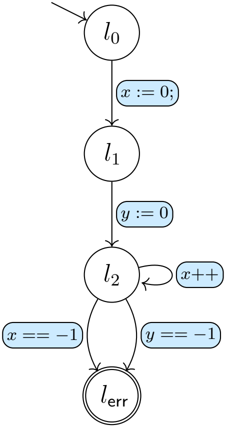


$\overline{\mathcal{A}_1}$:

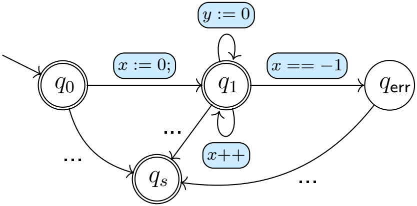


Trace Abstraction – Interpolant Based Approach

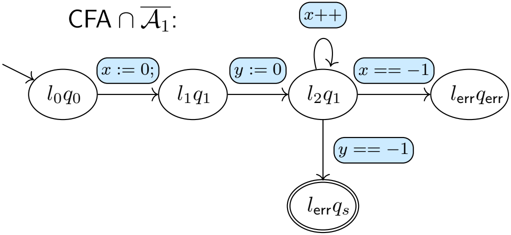
CFA:



$\overline{\mathcal{A}_1}$:



CFA $\cap \overline{\mathcal{A}_1}$:



Outline

- Motivation
- Example
- LTL Software Model Checking in CPAchecker
- Trace Abstraction
- **Outlook**

Outlook

- Implement *Trace Abstraction algorithm* for termination arguments
- Make use of *Adjustable Block Encoding* (ABE)
- ...

References

[1] B. Alpern and F. B. Schneider. Recognizing safety and liveness. Distributed Computing, 2(3):117–126, 1987.

[2] C. Baier and J. Katoen. Principles of model checking. MIT Press, 2008.

[3] D. Dietsch, M. Heizmann, V. Langenfeld, and A. Podelski. Fairness modulo theory: A new approach to LTL software model checking. In Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, volume 9206 of Lecture Notes in Computer Science, pages 49–66. Springer, 2015.

[4] M. Heizmann, J. Hoenicke, and A. Podelski. Refinement of trace abstraction. In Static Analysis, 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings, volume 5673 of Lecture Notes in Computer Science, pages 69–85. Springer, 2009.

[5] M. Heizmann, J. Hoenicke, and A. Podelski. Software model checking for people who love automata. In Proc. CAV, LNCS 8044, pages 36–52. Springer, 2013.