reachability analysis for continuous one counter automata

Lars Van Roy
dept. of Mathematics and Computer Science
University of Antwerp
lars.vanroy@student.uantwerpen.be

August 24, 2021

Problem overview

Problem overview I

- Reachability analysis

Problem overview I

- Reachability analysis
  - Are all conditional statements satisfiable

Problem overview I

- Reachability analysis
  - Are all conditional statements satisfiable
- Explore set of values that a counter can have

Problem overview I

- Reachability analysis
  - Are all conditional statements satisfiable
- Explore set of values that a counter can have
- Analysis of one-counter continuous automata

Problem overview I

- Reachability analysis
  - Are all conditional statements satisfiable
- Explore set of values that a counter can have
- Analysis of one-counter continuous automata
- Only applicable to simple functions

Problem overview II

- Only applicable to simple functions

Problem overview II

- Only applicable to simple functions
    - Supported comparisons: $\leq, <, ==, \neq, >, \geq$

Problem overview II

- Only applicable to simple functions
  - Supported comparisons: $\leq, <, ==, \neq, >, \geq$
  - Supported operations: $+, -, =$

Problem overview II

- Only applicable to simple functions
    - Supported comparisons: $\leq, <, ==, \neq, >, \geq$
    - Supported operations: $+, -, =$
    - Restrictions only apply for counters

Problem overview II

- Only applicable to simple functions
    - Supported comparisons: $\leq, <, ==, \neq, >, \geq$
    - Supported operations: $+, -, =$
    - Restrictions only apply for counters
    - Counter must be of type integer

Problem overview II

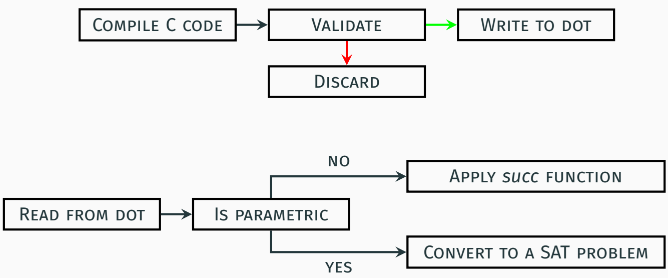
- Only applicable to simple functions
  - Supported comparisons: $\leq, <, ==, \neq, >, \geq$
  - Supported operations: $+, -, =$
  - Restrictions only apply for counters
  - Counter must be of type integer
  - Conditions and operations on counters can be performed with
    - Parameters
    - Constants

Problem overview II

- Only applicable to simple functions
  - Supported comparisons: $\leq, <, ==, \neq, >, \geq$
  - Supported operations: $+, -, =$
  - Restrictions only apply for counters
  - Counter must be of type integer
  - Conditions and operations on counters can be performed with
    - Parameters
    - Constants
  - There can only be one consecutive counter

Approach

Approach



Compile C code
Validate
Write to dot
Discard
Read from dot
Is parametric
no
Apply *succ* function
yes
Convert to a SAT problem

Reachability intervals

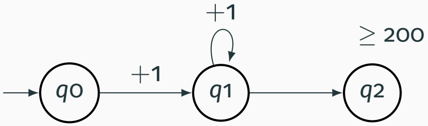
- Interval representing possible counter values

Reachability intervals

- Interval representing possible counter values
- Tracked for each of the nodes

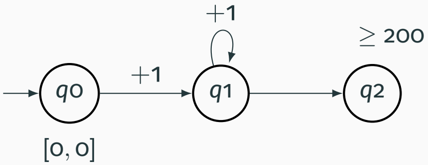
Reachability intervals

- Interval representing possible counter values
- Tracked for each of the nodes



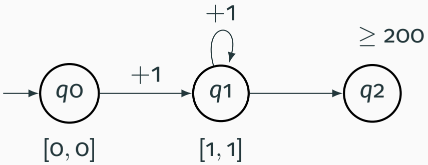
Reachability intervals

- Interval representing possible counter values
- Tracked for each of the nodes



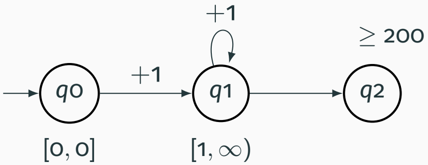
Reachability intervals

- Interval representing possible counter values
- Tracked for each of the nodes



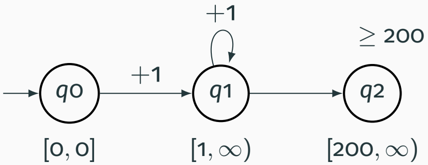
Reachability intervals

- Interval representing possible counter values
- Tracked for each of the nodes



Reachability intervals

- Interval representing possible counter values
- Tracked for each of the nodes



Successor function

Successor function I

- Approach for non-parametric automata

Successor function I

- Approach for non-parametric automata
- Evaluate the reachability intervals iteratively

Successor function I

- Approach for non-parametric automata
- Evaluate the reachability intervals iteratively
- Acceleration to prevent infinite loops

Successor function II

$$\begin{aligned} succ_i(p,q) := & \bigcup \{(R_i(p,q) + (0,z]) \cap \tau(q) \mid (p,z,q) \in T, z > 0\} \\ & \cup \bigcup \{(R_i(p,q) + [z,0)) \cap \tau(q) \mid (p,z,q) \in T, z < 0\} \\ & \cup \bigcup \{(R_i(p,q) + [0,0]) \cap \tau(q) \mid (p,0,q) \in T, z = 0\} \end{aligned}$$

1. Generate the next interval for the edge going from p to q
2. Apply the edge operation to the current interval
3. Ensure that the interval is within the node bounds

Successor function II

$$
\begin{aligned}
succ_i(p,q) := &\bigcup\{(R_i(p,q) + (0,z]) \cap \tau(q) \mid (p,z,q) \in T, z > 0\} \\
\cup &\bigcup\{(R_i(p,q) + [z,0)) \cap \tau(q) \mid (p,z,q) \in T, z < 0\} \\
\cup &\bigcup\{(R_i(p,q) + [0,0]) \cap \tau(q) \mid (p,0,q) \in T, z = 0\}
\end{aligned}
$$

1. Generate the next interval for the edge going from p to q
2. Add the operation interval to the current reachability interval
   - operation interval = (0, 1] * z
3. Ensure that the interval is within the node bounds

Successor function II

$$
\begin{aligned}
succ_i(p,q) := \bigcup & \boxed{\{(R_i(p,q) + (0,z])} \cap \tau(q) \mid (p,z,q) \in T, z > 0\} \\
\cup \bigcup & \boxed{\{(R_i(p,q) + [z,0))} \cap \tau(q) \mid (p,z,q) \in T, z < 0\} \\
\cup \bigcup & \boxed{\{(R_i(p,q) + [0,0])} \cap \tau(q) \mid (p,0,q) \in T, z = 0\}
\end{aligned}
$$

1. Generate the next interval for the edge going from p to q
2. Add the operation interval to the current reachability interval
    - operation interval = (0, 1] * z
3. Ensure that the interval is within the node bounds

Successor function II

$$
\begin{aligned}
succ_i(p,q) := & \bigcup \{ (R_i(p,q) + (0,z]) \cap \tau(q) \mid (p,z,q) \in T, z > 0 \} \\
& \cup \bigcup \{ (R_i(p,q) + [z,0)) \cap \tau(q) \mid (p,z,q) \in T, z < 0 \} \\
& \cup \bigcup \{ (R_i(p,q) + [0,0]) \cap \tau(q) \mid (p,0,q) \in T, z = 0 \}
\end{aligned}
$$

1. Generate the next interval for the edge going from p to q
2. Add the operation interval to the current reachability interval
    - operation interval = (0, 1] * z
3. Ensure that the interval is within the node bounds

Successor function III

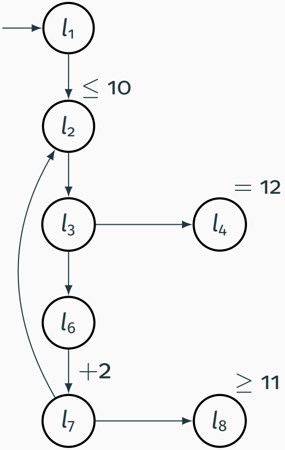
- Iteratively update intervals until no more changes occur

Successor function III

- Iteratively update intervals until no more changes occur
- Only apply the *succ* function to non-empty intervals

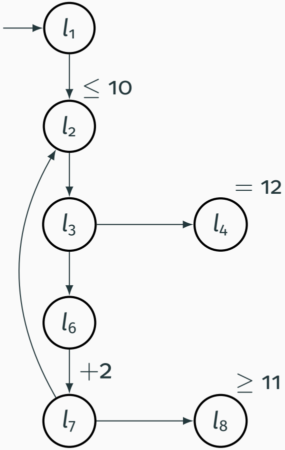
Successor function IV

```c
int func() {
    for (int i;i < 11;) {
        if (i == 12) {
            return -1;
        }
        i += 2;
    }
    return 0;
}
```



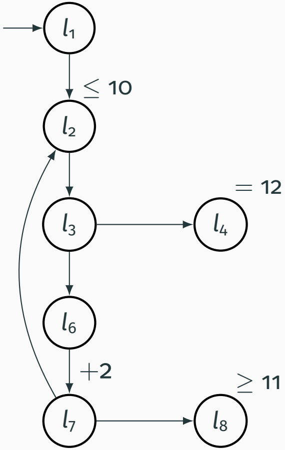
Successor function V

p	q	R_0
l_1	l_2	$[0, 0]$
l_2	l_3	$\emptyset$
l_3	l_4	$\emptyset$
l_3	l_6	$\emptyset$
l_6	l_7	$\emptyset$
l_7	l_2	$\emptyset$
l_7	l_8	$\emptyset$



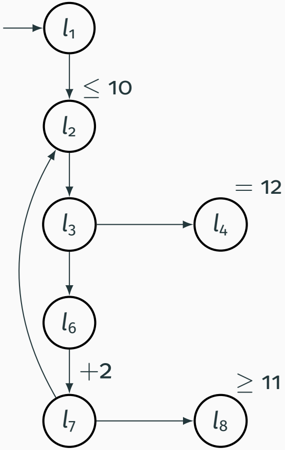
Successor function V

p	q	R_0	R_1
l_1	l_2	$[0, 0]$	$[0, 0]$
l_2	l_3	$\emptyset$	$[0, 0]$
l_3	l_4	$\emptyset$	$\emptyset$
l_3	l_6	$\emptyset$	$\emptyset$
l_6	l_7	$\emptyset$	$\emptyset$
l_7	l_2	$\emptyset$	$\emptyset$
l_7	l_8	$\emptyset$	$\emptyset$



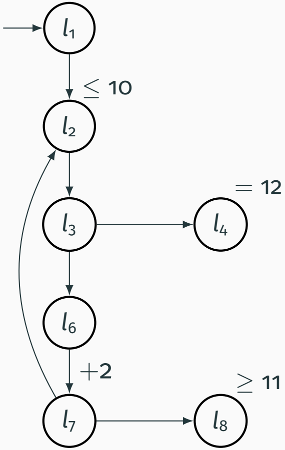
Successor function V

p	q	R_0	R_1	R_2
l_1	l_2	[0, 0]	[0, 0]	[0, 0]
l_2	l_3	$\emptyset$	[0, 0]	[0, 0]
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	$\emptyset$	$\emptyset$	[0, 0]
l_6	l_7	$\emptyset$	$\emptyset$	$\emptyset$
l_7	l_2	$\emptyset$	$\emptyset$	$\emptyset$
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$



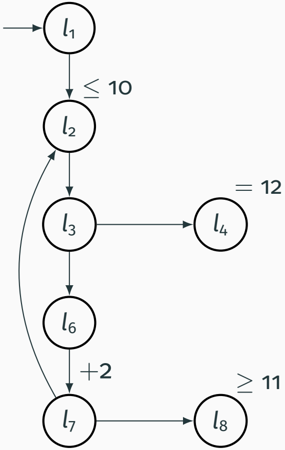
Successor function V

p	q	R_0	R_1	R_2	R_3
l_1	l_2	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_2	l_3	$\emptyset$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	$\emptyset$	$\emptyset$	$[0, 0]$	$[0, 0]$
l_6	l_7	$\emptyset$	$\emptyset$	$\emptyset$	$(0, 2]$
l_7	l_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Successor function V

p	q	R_0	R_1	R_2	R_3	R_4
l_1	l_2	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_2	l_3	$\emptyset$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	$\emptyset$	$\emptyset$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_6	l_7	$\emptyset$	$\emptyset$	$\emptyset$	$(0, 2]$	$(0, 2]$
l_7	l_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$(0, 2]$
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Acceleration

Acceleration I

- Accelerate by moving bounds of an interval closer to fix point

Acceleration I

- Accelerate by moving bounds of an interval closer to fix point
- The full loop must be discovered

Acceleration I

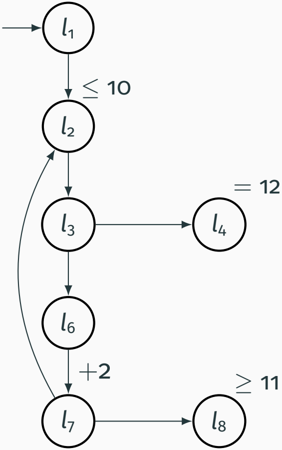
- Accelerate by moving bounds of an interval closer to fix point
- The full loop must be discovered
- Select interval closest to its bound

Acceleration I

- Accelerate by moving bounds of an interval closer to fix point
- The full loop must be discovered
- Select interval closest to its bound
- Set interval bound equal to the node/automaton bound

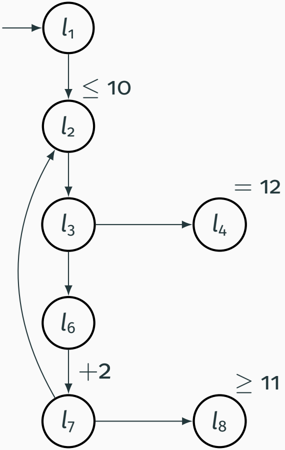
Acceleration II

p	q	R_4
l_1	l_2	$[0, 0]$
l_2	l_3	$[0, 0]$
l_3	l_4	$\emptyset$
l_3	l_6	$[0, 0]$
l_6	l_7	$(0, 2]$
l_7	l_2	$(0, 2]$
l_7	l_8	$\emptyset$



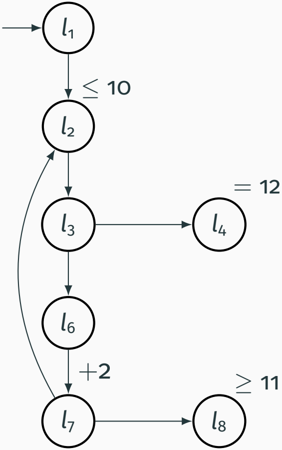
Acceleration II

p	q	R_4	R_5
l_1	l_2	$[0, 0]$	$[0, 0]$
l_2	l_3	$[0, 0]$	$[0, 0]$
l_3	l_4	$\emptyset$	$\emptyset$
l_3	l_6	$[0, 0]$	$[0, 0]$
l_6	l_7	$(0, 2]$	$(0, 2]$
l_7	l_2	$(0, 2]$	$(0, 10]$
l_7	l_8	$\emptyset$	$\emptyset$



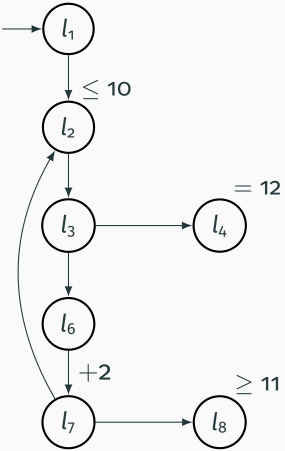
Acceleration II

p	q	R_4	R_5	R_6
l_1	l_2	[0, 0]	[0, 0]	[0, 0]
l_2	l_3	[0, 0]	[0, 0]	(0, 10]
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	[0, 0]	[0, 0]	[0, 0]
l_6	l_7	(0, 2]	(0, 2]	(0, 2]
l_7	l_2	(0, 2]	(0, 10]	(0, 10]
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$



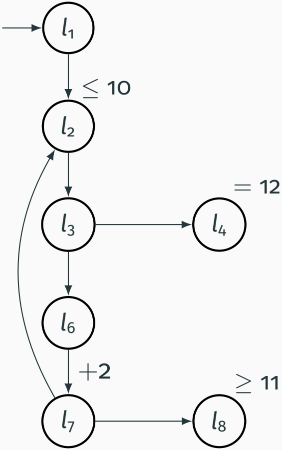
Acceleration II

p	q	R_4	R_5	R_6	R_7
l_1	l_2	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_2	l_3	$[0, 0]$	$[0, 0]$	$(0, 10]$	$(0, 10]$
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	$[0, 0]$	$[0, 0]$	$[0, 0]$	$(0, 10]$
l_6	l_7	$(0, 2]$	$(0, 2]$	$(0, 2]$	$(0, 2]$
l_7	l_2	$(0, 2]$	$(0, 10]$	$(0, 10]$	$(0, 10]$
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



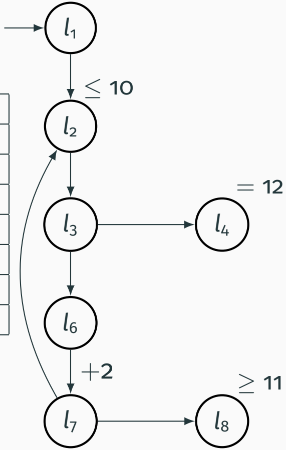
Acceleration II

p	q	R_4	R_5	R_6	R_7	R_8
l_1	l_2	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_2	l_3	$[0, 0]$	$[0, 0]$	$(0, 10]$	$(0, 10]$	$(0, 10]$
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	$[0, 0]$	$[0, 0]$	$[0, 0]$	$(0, 10]$	$(0, 10]$
l_6	l_7	$(0, 2]$	$(0, 2]$	$(0, 2]$	$(0, 2]$	$(0, 12]$
l_7	l_2	$(0, 2]$	$(0, 10]$	$(0, 10]$	$(0, 10]$	$(0, 10]$
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



Acceleration II

p	q	R_5	R_6	R_7	R_8	R_9
l_1	l_2	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$	$[0, 0]$
l_2	l_3	$[0, 0]$	$[0, 10]$	$(0, 10]$	$(0, 10]$	$(0, 10]$
l_3	l_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
l_3	l_6	$[0, 0]$	$[0, 0]$	$[0, 10]$	$(0, 10]$	$(0, 10]$
l_6	l_7	$(0, 2]$	$(0, 2]$	$(0, 2]$	$(0, 12]$	$(0, 12]$
l_7	l_2	$(0, 10]$	$(0, 10]$	$(0, 10]$	$(0, 10]$	$(0, 10]$
l_7	l_8	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$[11, 12]$



Shortcomings

- Does not work with variables

Shortcomings

- Does not work with variables
- Can result in false positives

SAT problem

SAT problem I

- Convert to a satisfiability problem

SAT problem I

- Convert to a satisfiability problem
- Can handle variables

SAT problem I

- Convert to a satisfiability problem
- Can handle variables
- Try to guess rather than compute

SAT problem II

- conditions

SAT problem II

- conditions
    - The initial interval needs to be $[0, 0]$

SAT problem II

- conditions
  - The initial interval needs to be $[0, 0]$
  - The successor function needs to be applicable to all intervals

SAT problem II

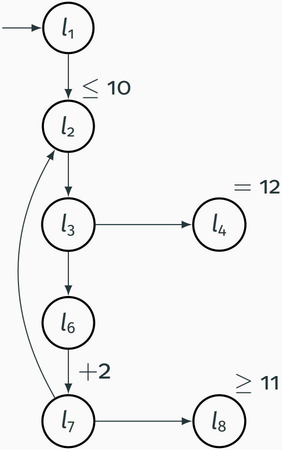
- conditions
  - The initial interval needs to be $[0, 0]$
  - The successor function needs to be applicable to all intervals
  - Loops will be self satisfying

SAT problem II

- conditions
  - The initial interval needs to be $[0, 0]$
  - The successor function needs to be applicable to all intervals
  - Loops will be self satisfying
    - One predecessor not part of loop

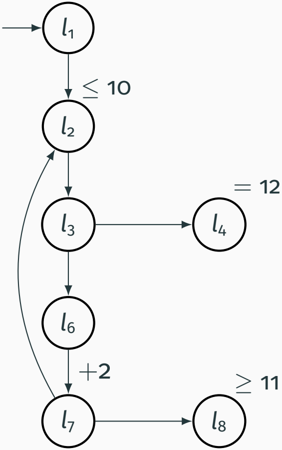
SAT problem III

p	q	configuration
l_1	l_2	$[0, 0]$
l_2	l_3	$\emptyset$
l_3	l_4	$\emptyset$
l_3	l_6	$\emptyset$
l_6	l_7	$\emptyset$
l_7	l_2	$\emptyset$
l_7	l_8	$\emptyset$



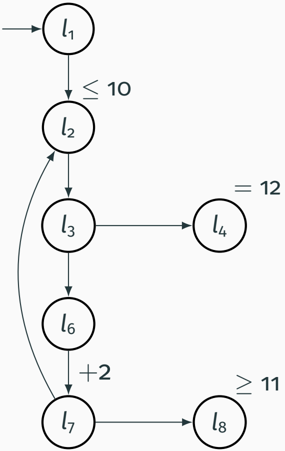
SAT problem III

p	q	configuration
l_1	l_2	$[0, 0]$
l_2	l_3	$[0, 0]$
l_3	l_4	$\emptyset$
l_3	l_6	$[0, 0]$
l_6	l_7	$(0, 2]$
l_7	l_2	$(0, 2]$
l_7	l_8	$\emptyset$



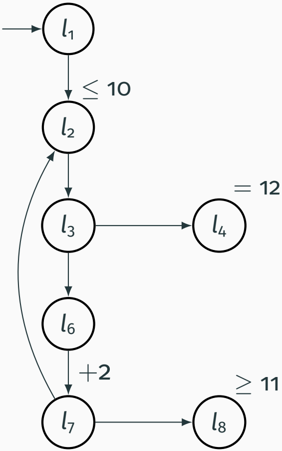
SAT problem III

p	q	configuration
l_1	l_2	$[0, 0]$
l_2	l_3	$(1, 10]$
l_3	l_4	$\emptyset$
l_3	l_6	$(1, 10]$
l_6	l_7	$(1, 12]$
l_7	l_2	$(1, 12]$
l_7	l_8	$[11, 12]$



SAT problem III

p	q	configuration
l_1	l_2	$[0, 0]$
l_2	l_3	$(0, 10]$
l_3	l_4	$\emptyset$
l_3	l_6	$(0, 10]$
l_6	l_7	$(0, 12]$
l_7	l_2	$(0, 12]$
l_7	l_8	$[11, 12]$



SAT problem IV

- Requires a large amount of variables

SAT problem IV

- Requires a large amount of variables
  - Four variables per interval

SAT problem IV

- Requires a large amount of variables
  - Four variables per interval
  - Two variables per node condition

SAT problem IV

- Requires a large amount of variables
  - Four variables per interval
  - Two variables per node condition
  - Three variables per edge

SAT problem IV

- Requires a large amount of variables
  - Four variables per interval
  - Two variables per node condition
  - Three variables per edge
- Slower than the first approach

SAT problem IV

- Requires a large amount of variables
  - Four variables per interval
  - Two variables per node condition
  - Three variables per edge
- Slower than the first approach
- Can give false positives

SAT problem IV

- Requires a large amount of variables
  - Four variables per interval
  - Two variables per node condition
  - Three variables per edge
- Slower than the first approach
- Can give false positives
- Only use SAT in case parameters are present

Use case: Xrdp

Use case: Xrdp

- Graphical login to remote machines using RDP

Use case: Xrdp

- Graphical login to remote machines using RDP
- Medium-sized C project
    - 94 458 lines of code
    - 1 328 functions

Use case: Xrdp

- Graphical login to remote machines using RDP
- Medium-sized C project
  - 94 458 lines of code
  - 1 328 functions
- Analysed 73.42% of the functions

Use case: Xrdp

- Graphical login to remote machines using RDP
- Medium-sized C project
    - 94 458 lines of code
    - 1 328 functions
- Analysed 73.42% of the functions
- 33 Convertible functions

Use case: Xrdp

- Graphical login to remote machines using RDP
- Medium-sized C project
    - 94 458 lines of code
    - 1 328 functions
- Analysed 73.42% of the functions
- 33 Convertible functions
- 2 Parametric automata

Manual verification

Manual verification

- Introduce dead code in xrdp

Manual verification

- Introduce dead code in xrdp
- 7 different types of dead code

Manual verification

- Introduce dead code in xrdp
- 7 different types of dead code
- Test suite with 115 tests

Conclusion

Conclusion

- The proposed approach is capable of identifying dead code

Conclusion

- The proposed approach is capable of identifying dead code
- No false negatives

Conclusion

- The proposed approach is capable of identifying dead code
- No false negatives
- Need for a different compiler

Conclusion

- The proposed approach is capable of identifying dead code
- No false negatives
- Need for a different compiler
- Further optimizations to the code constraints should be considered