Software Model Checking via Large-Block Encoding

Dirk Beyer[1], Alessandro Cimatti[2], Alberto Griggio[3], Erkan Keremoglu[1], Roberto Sebastiani[3]

[1]SFU, Vancouver, Canada, [2]FBK-IRST and [3]DISI-Univ. of Trento, Italy

Motivations

- SMT: very promising technology for verification
  - SMT solvers: efficient, powerful, scalable
  - Several SMT-based verification techniques recently proposed
- Software Model Checking: effective technique for software verification (e.g. SLAM, BLAST, verification of device drivers)
  - Popular approach: lazy abstraction with analysis of an abstract reachability tree (ART)
- Current ART-based approaches do not take full advantage of SMT solvers
  - Explicit exploration of the ART, SMT only used (mostly) for conjunctions of constraints

Contribution

- Large-Block Encoding: (simple) generalization of traditional ART-based approach aimed at better exploiting SMT technology
  - Less explicit search on the ART, more symbolic search within the SMT solver
  - Empirical evidence of the benefits on a set of standard benchmark C programs

Outline

- Background
- Large-Block Encoding
- Experimental evaluation

Background – Programs and CFAs

Programs represented as control-flow automata (CFAs)

- A CFA is a pair (L, G), where:
  - L: set of program locations
  - G: set of edges $L \times Op \times L$
- l_0 entry point of a program,

  l_E error location

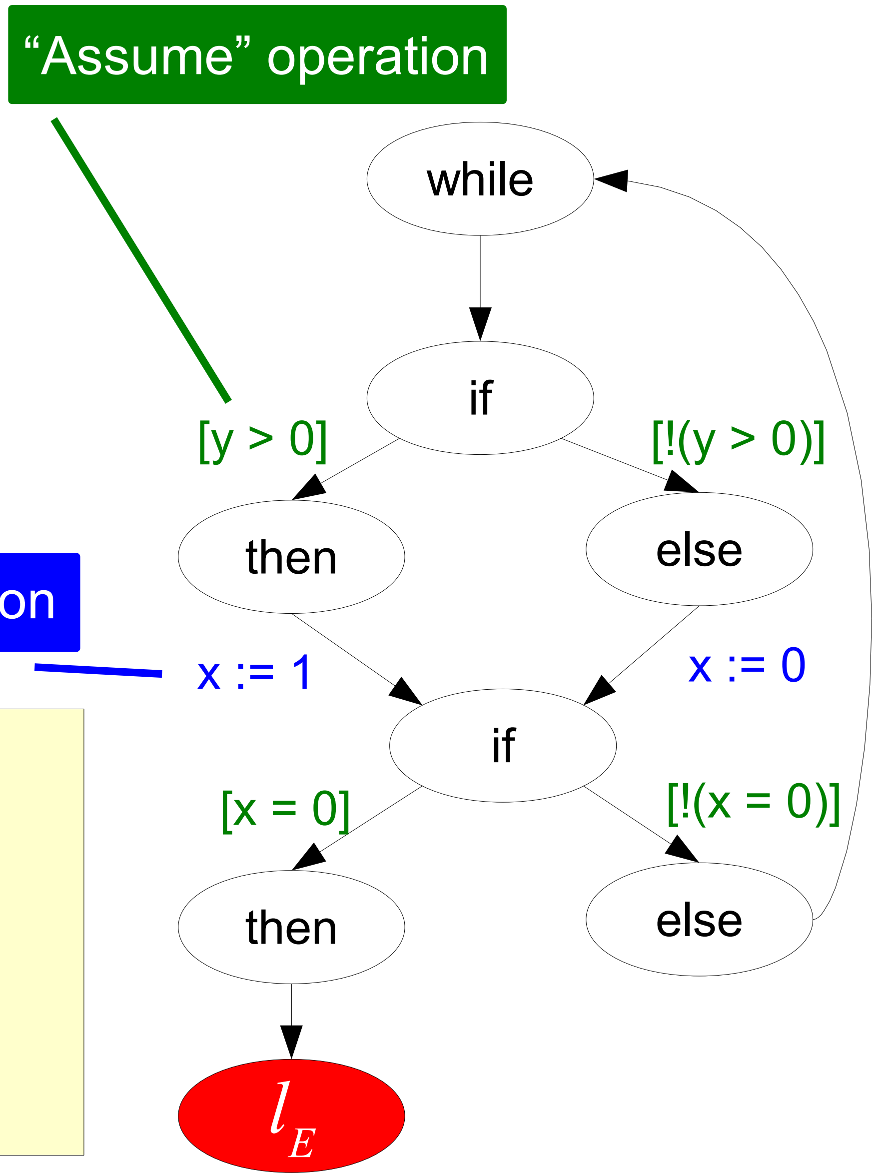
Background – Programs and CFAs

Programs represented as control-flow automata (CFAs)

- A CFA is a pair *(L, G)*, where:
  - L: set of program locations
  - G: set of edges $L \times Op \times L$
- l_0 entry point of a program,

  l_E error location

“Assign” operation



- Example:

```
while (1) {
   if (y > 0) x = 1;
   else x = 0;
   if (x == 0) goto ERROR;
}
```

Background – semantics

- Concrete state of a program: (l, s), where
  - l is a location
  - s is an assignment to program variables (a formula $\bigwedge_i x_i = v_i$)
- Concrete semantics of an operation op given by SP_{op}:
  - Assign: $\mathsf{SP}_{x:=e}(\varphi) = \exists \hat{x} : \varphi_{[x \mapsto \hat{x}]} \wedge (x = e_{[x \mapsto \hat{x}]})$
  - Assume: $\mathsf{SP}_p(\varphi) = \varphi \wedge p$
- Path: seq. $\sigma = \langle (op_1, l_1) \ldots (op_n, l_n) \rangle$ where (l_{i-1}, op_i, l_i) is a CFA edge
  - Semantics $\mathsf{SP}_\sigma(\varphi) = \mathsf{SP}_{op_n}(\ldots \mathsf{SP}_{op_1}(\varphi) \ldots)$
  - Feasible if $\mathsf{SP}_\sigma(true)$ is satisfiable
- Location l reachable iff exists feasible path $\langle (op_1, l_1) \ldots (op_n, l) \rangle$
- Program safe iff l_E not reachable

Background – ART-based SW MC

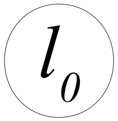
- Abstraction of φ : $\alpha(\varphi)$ such that $\varphi \models \alpha(\varphi)$
  - Abstract SP: $\mathsf{SP}^{\alpha}_{op}(\varphi) = \alpha(\mathsf{SP}_{op}(\varphi))$

- Abstract Reachability Tree (ART): unwinding of the CFA in an abstract space
  - Each node is an abstract state (l, φ)
  - children of (l, φ): abstract successors: $\{(l_i, \hat{\varphi}_i)\}_i$
    - (l, op_i, l_i) is an edge in the CFA
    - $\hat{\varphi}_i = \mathsf{SP}^{\alpha}_{op_i}(\varphi)$ and $\hat{\varphi}_i \not\models \bot$
  - (l, φ) has children only if not covered
    - there is no (l, ψ) in the ART s.t. $\varphi \models \psi$
  - ART safe if there is no $(l_E, \cdot)$

Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

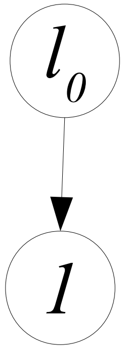
- Pick node



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

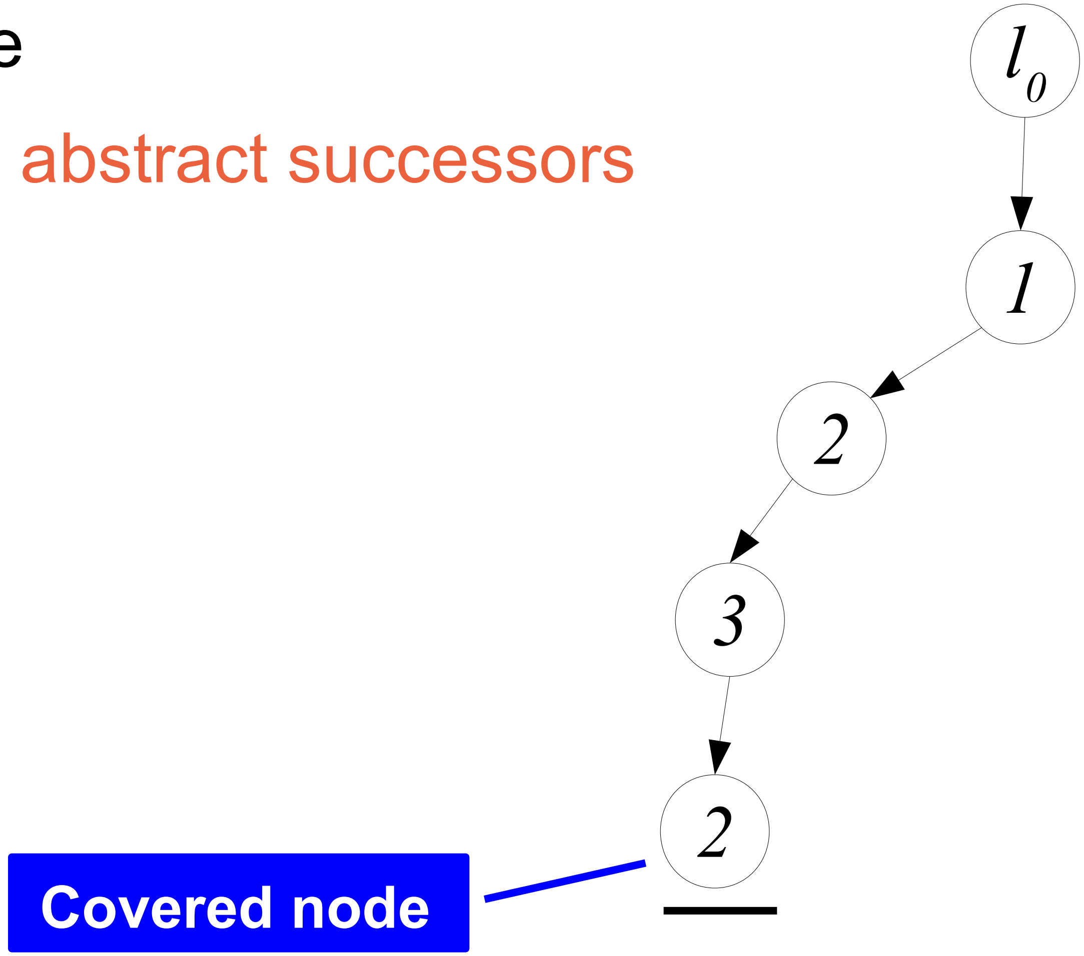
- Pick node
- Compute abstract successors



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

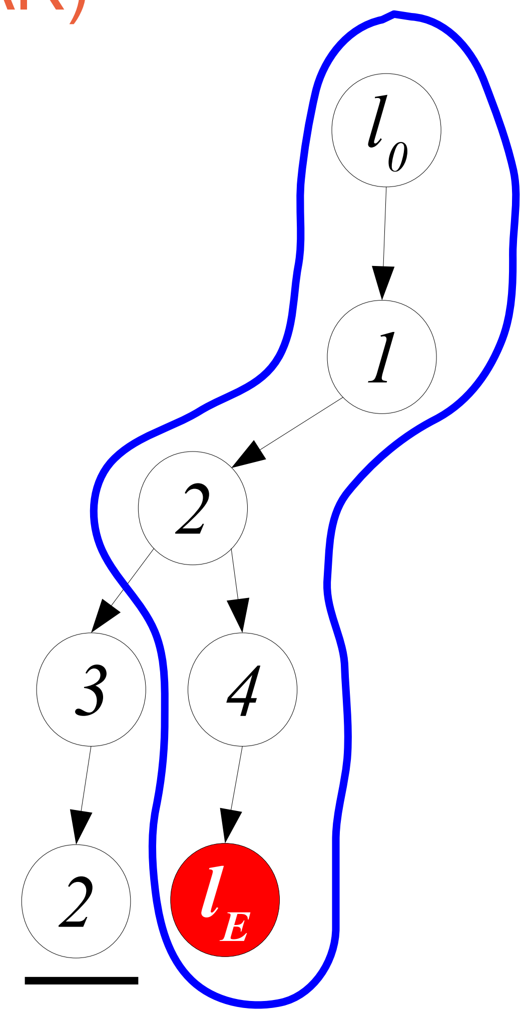
- Pick node
- Compute abstract successors



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

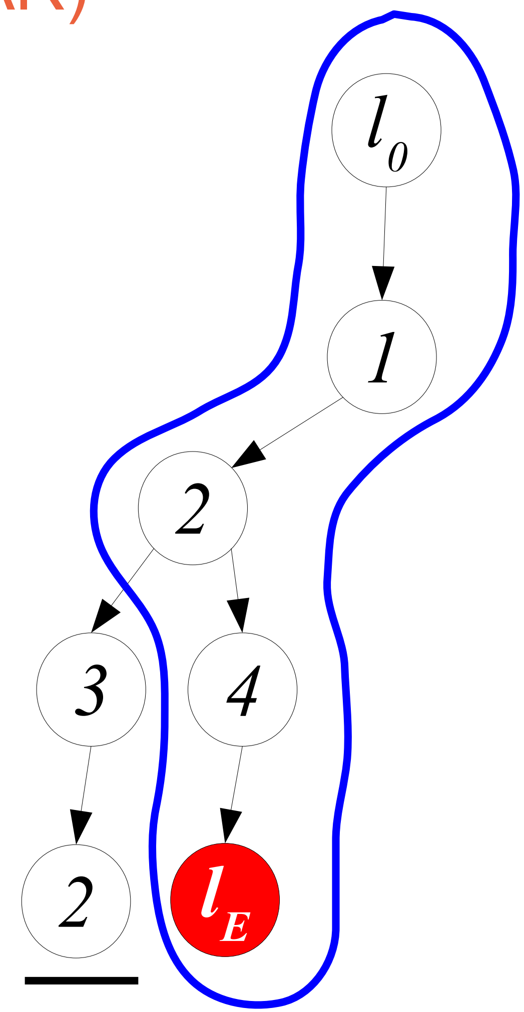
- Pick node
- Compute abstract successors
- If error reached:
  analyze abstract trace
  - If spurious:



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

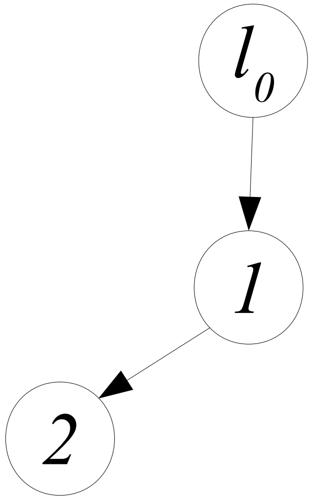
- Pick node
- Compute abstract successors
- If error reached:
  analyze abstract trace
  - If spurious:
    - refine abstraction



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

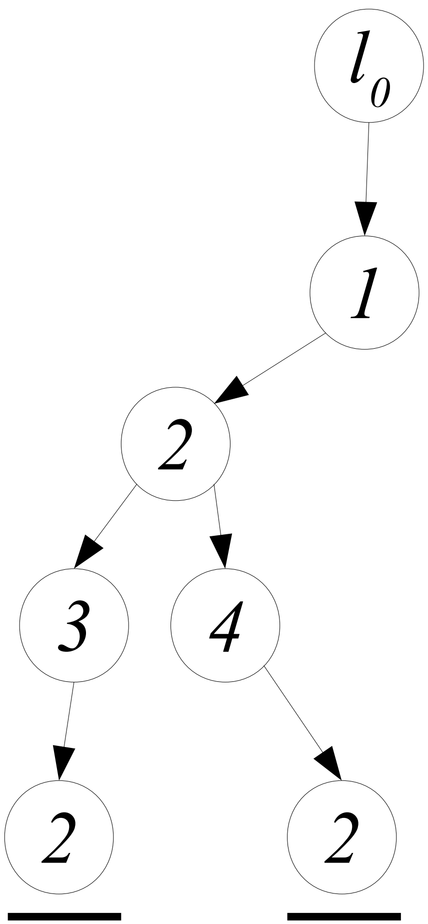
- Pick node
- Compute abstract successors
- If error reached:
  analyze abstract trace
  - If spurious:
    - refine abstraction
    - undo part of the ART



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

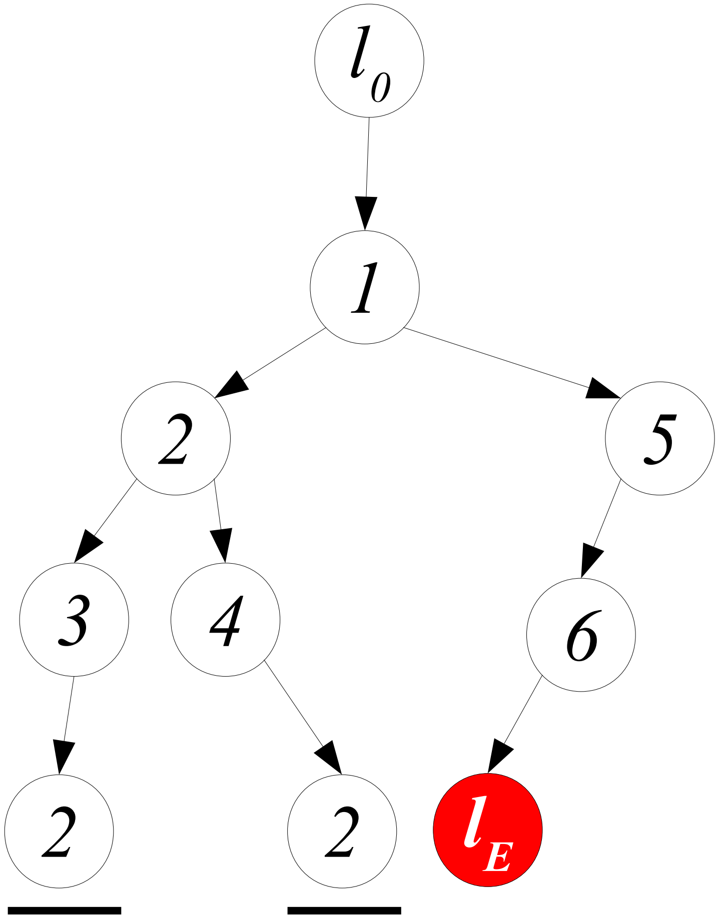
- Pick node
- Compute abstract successors
- If error reached: analyze abstract trace
  - If spurious:
    - refine abstraction
    - undo part of the ART
    - Rebuild subtree



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

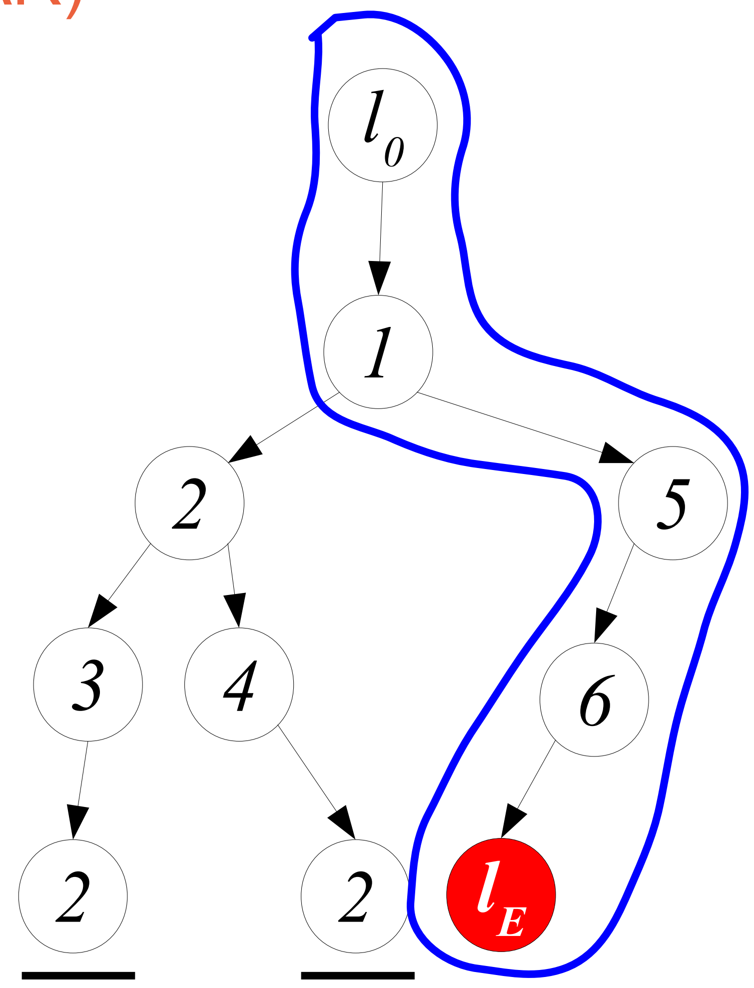
- Pick node
- Compute abstract successors
- If error reached: analyze abstract trace
  - If spurious:
    - refine abstraction
    - undo part of the ART
    - Rebuild subtree



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

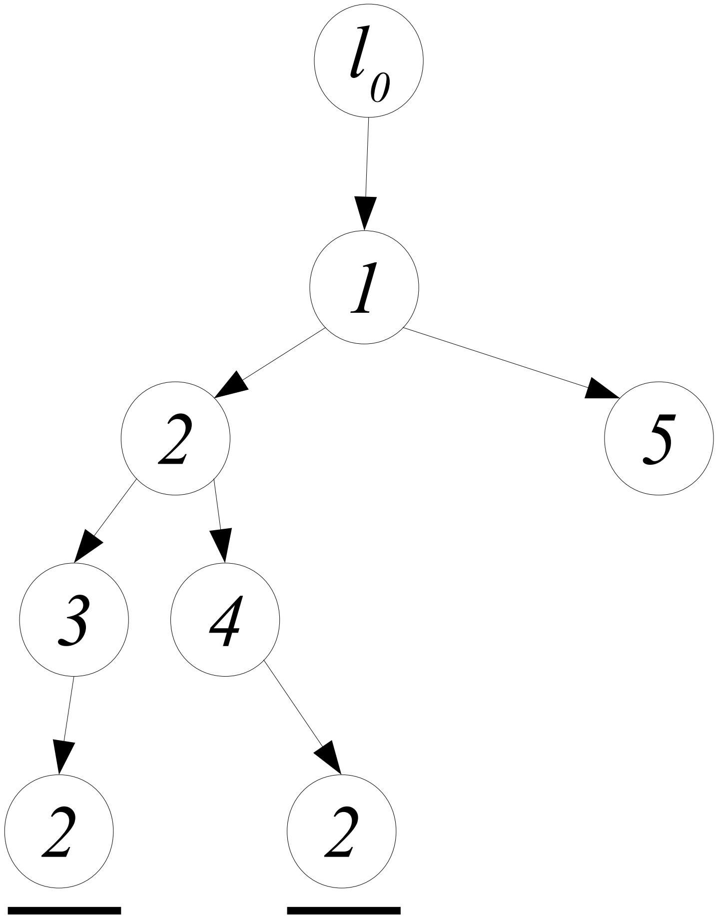
- Pick node
- Compute abstract successors
- If error reached: analyze abstract trace
  - If spurious:
    - refine abstraction
    - undo part of the ART
    - Rebuild subtree



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

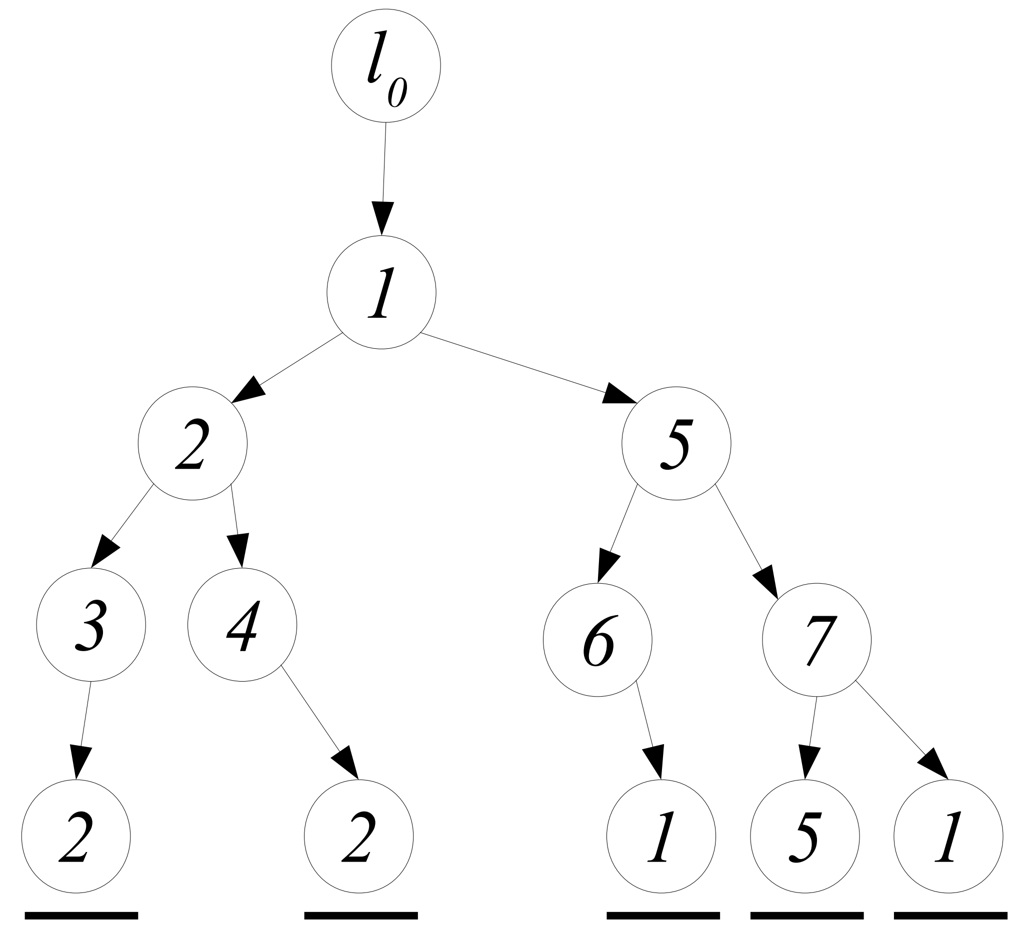
- Pick node
- Compute abstract successors
- If error reached:
  analyze abstract trace
  - If spurious:
    - refine abstraction
    - undo part of the ART
    - Rebuild subtree



Background – ART-based SW MC (2)

On-the-fly ART construction with counterexample-guided abstraction refinement (CEGAR)

- Pick node
- Compute abstract successors
- If error reached:
  analyze abstract trace
  - If spurious:
    - refine abstraction
    - undo part of the ART
    - Rebuild subtree
- ART safe ⇒ Program safe



Background – Predicate abstraction

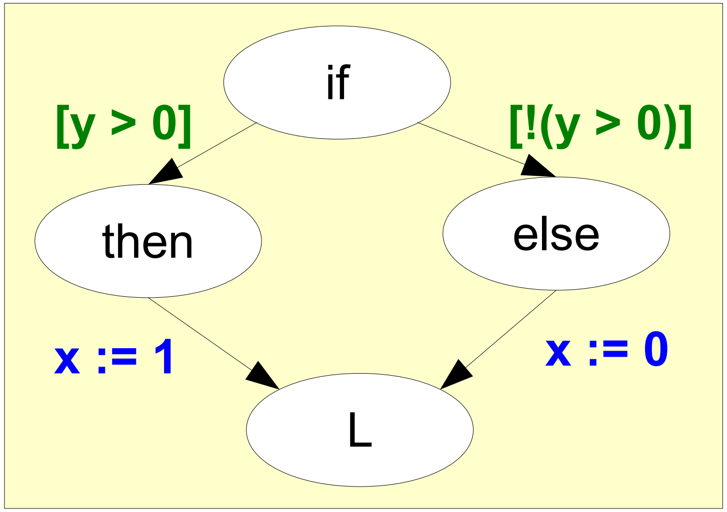
- Abstraction of φ as a Boolean combination of a set of predicates P
- Boolean abstraction
  - $\alpha_P^{\mathbb{B}}(\varphi)$ strongest Boolean combination of P s.t. $\varphi \models \alpha^{\mathbb{B}}(\varphi)$
  - Expensive to compute
    - Traditional approach: $2^{|P|}$ calls to a decision procedure
- Cartesian abstraction
  - $\alpha_P^{\mathbb{C}}(\varphi) = \bigwedge\{p \in P \mid \varphi \models p\} \cup \bigwedge\{\neg p \mid p \in P \text{ and } \varphi \models \neg p\}$
  - Much cheaper to compute
  - Much weaker
- Abstraction refinement: add more predicates to P

Background – Cartesian vs Boolean abst.

- Example

$$\varphi = ((y > 0) \wedge (x = 1)) \vee (\neg(y > 0) \wedge (x = 0))$$

$$P = \{(x = 0), (y = 0)\}$$



Background – Cartesian vs Boolean abst.

- Example

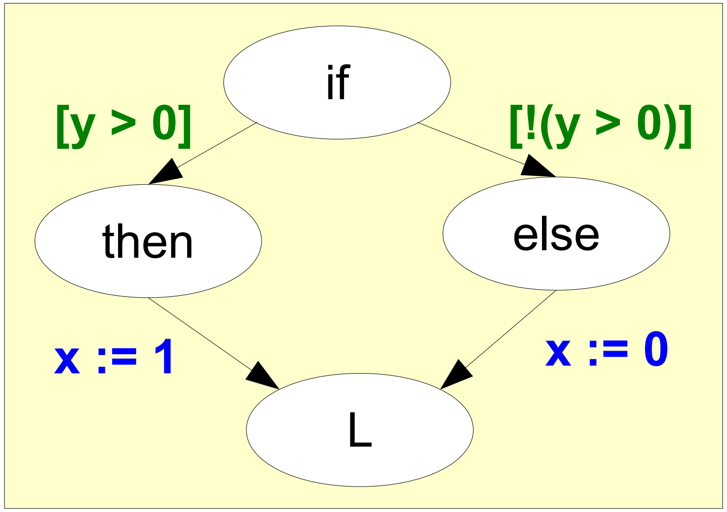
$$\varphi = ((y > 0) \wedge (x = 1)) \vee (\neg(y > 0) \wedge (x = 0))$$

$$P = \{(x = 0), (y = 0)\}$$

- $\alpha_P^{\mathbb{C}}(\varphi) = \top$, since

$$\varphi \not\models (x = 0) \qquad \varphi \not\models \neg(x = 0)$$

$$\varphi \not\models (y = 0) \qquad \varphi \not\models \neg(y = 0)$$



Background – Cartesian vs Boolean abst.

- Example

$$\varphi = ((y > 0) \wedge (x = 1)) \vee (\neg(y > 0) \wedge (x = 0))$$

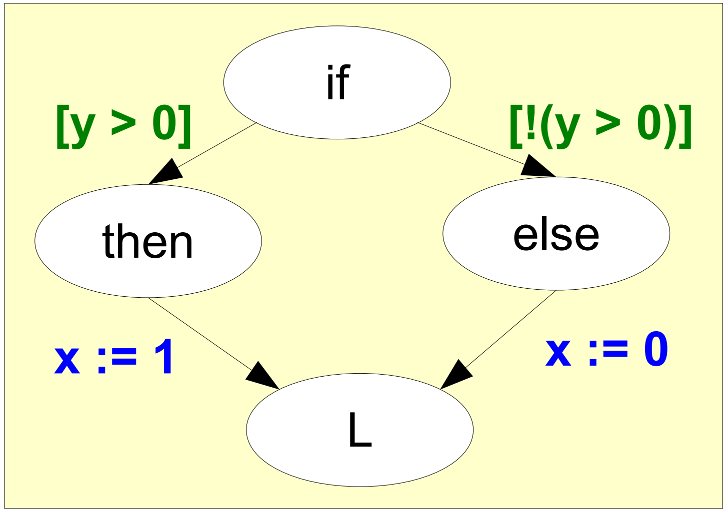
$$P = \{(x = 0), (y = 0)\}$$

- $\alpha_P^{\mathbb{C}}(\varphi) = \top$, since

$$\varphi \not\models (x = 0) \quad \varphi \not\models \neg(x = 0)$$

$$\varphi \not\models (y = 0) \quad \varphi \not\models \neg(y = 0)$$

- $\alpha_P^{\mathbb{B}}(\varphi) = (y = 0) \rightarrow (x = 0)$

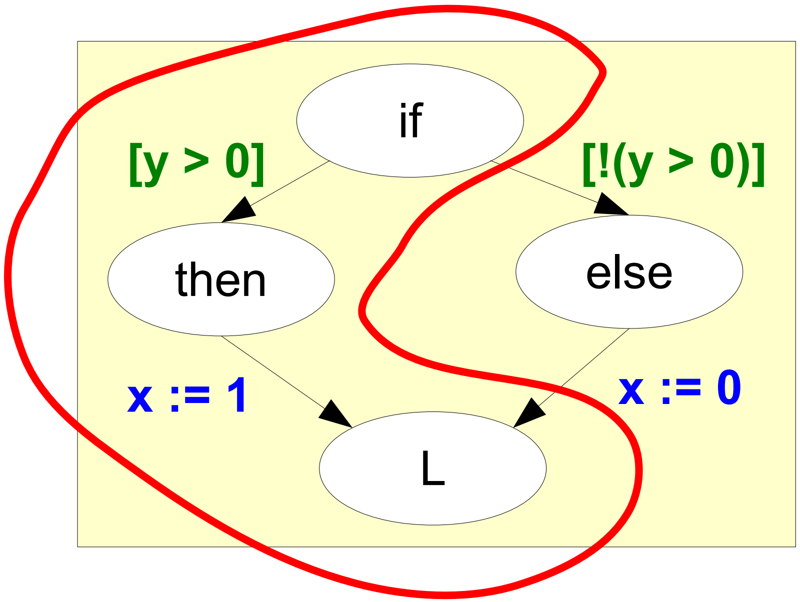


Background – Cartesian vs Boolean abst.

- Example

$$\varphi = ((y > 0) \wedge (x = 1)) \vee (\neg(y > 0) \wedge (x = 0))$$

$$P = \{(x = 0), (y = 0)\}$$



- $\alpha_P^{\mathbb{C}}(\varphi) = \top$, since

$$\varphi \not\models (x = 0) \quad \varphi \not\models \neg(x = 0)$$
$$\varphi \not\models (y = 0) \quad \varphi \not\models \neg(y = 0)$$

- $\alpha_P^{\mathbb{B}}(\varphi) = (y = 0) \rightarrow (x = 0)$

- However, e.g. for $\varphi_{then} = (y > 0) \wedge (x = 1)$

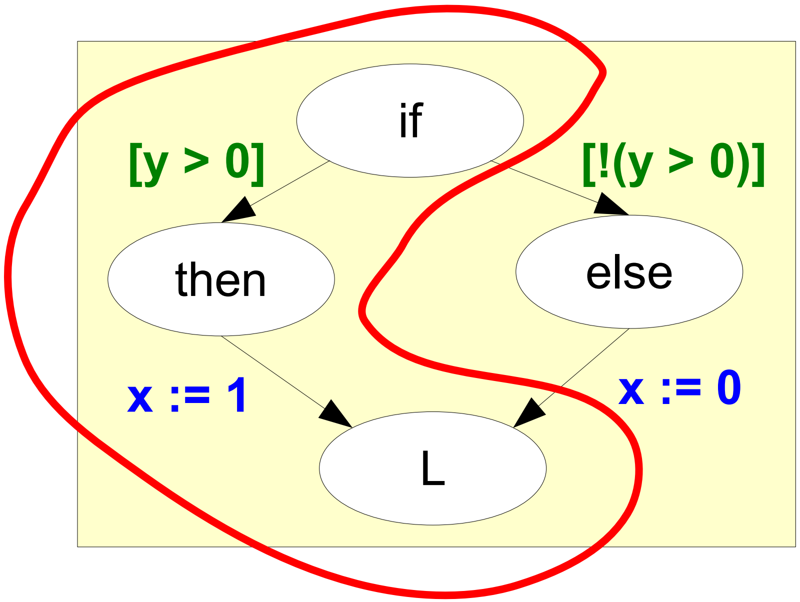
$$\alpha_P^{\mathbb{C}}(\varphi_{then}) = \alpha_P^{\mathbb{B}}(\varphi_{then}) = \neg(x = 0) \wedge \neg(y = 0)$$

Background – Cartesian vs Boolean abst.

- Example

$$\varphi = ((y > 0) \wedge (x = 1)) \vee (\neg(y > 0) \wedge (x = 0))$$

$$P = \{(x = 0), (y = 0)\}$$



- $\alpha_P^{\mathbb{C}}(\varphi) = \top$, since

$$\varphi \not\models (x = 0) \quad \varphi \not\models \neg(x = 0)$$
$$\varphi \not\models (y = 0) \quad \varphi \not\models \neg(y = 0)$$

- $\alpha_P^{\mathbb{B}}(\varphi) = (y = 0) \rightarrow (x = 0)$

- However, e.g. for $\varphi_{then} = (y > 0) \wedge (x = 1)$

$$\alpha_P^{\mathbb{C}}(\varphi_{then}) = \alpha_P^{\mathbb{B}}(\varphi_{then}) = \neg(x = 0) \wedge \neg(y = 0)$$

Cart. abst. often good enough for ART-based SW MC in practice

Outline

- Background
- Large-Block Encoding
- Experimental evaluation

ART and SMT solvers

- Using an ART, reduced cost in computing abstractions
  - Separate $\mathrm{SP}_{op}^{\alpha}$ computation for each edge (*single block*)
  - Cartesian abstraction works well in practice
  - *Consequence: very simple queries to the SMT solver*

- However, up to exponential number of paths to explore *explicitly* in the ART
  - *Exponentially-many trivial SMT solver calls*

Power, scalability and features of modern SMT solvers not fully exploited

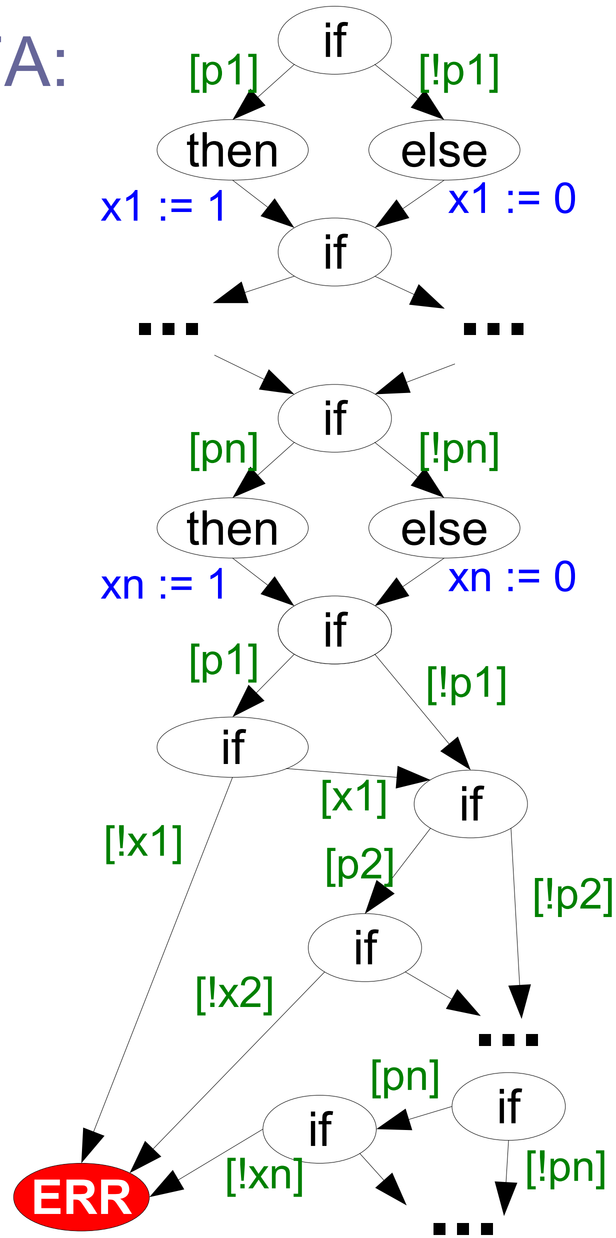
Example of exponential ART

```
if (p1) {x1 = 1;} else {x1 = 0;}
...
if (pn) {xn = 1;} else {xn = 0;}

if (p1) { if (!x1) goto ERROR; }
...
if (pn) { if (!xn) goto ERROR; }
```

Example of exponential ART

```
if (p1) {x1 = 1;} else {x1 = 0;}
...
if (pn) {xn = 1;} else {xn = 0;}

if (p1) { if (!x1) goto ERROR; }
...
if (pn) { if (!xn) goto ERROR; }
```

CFA:

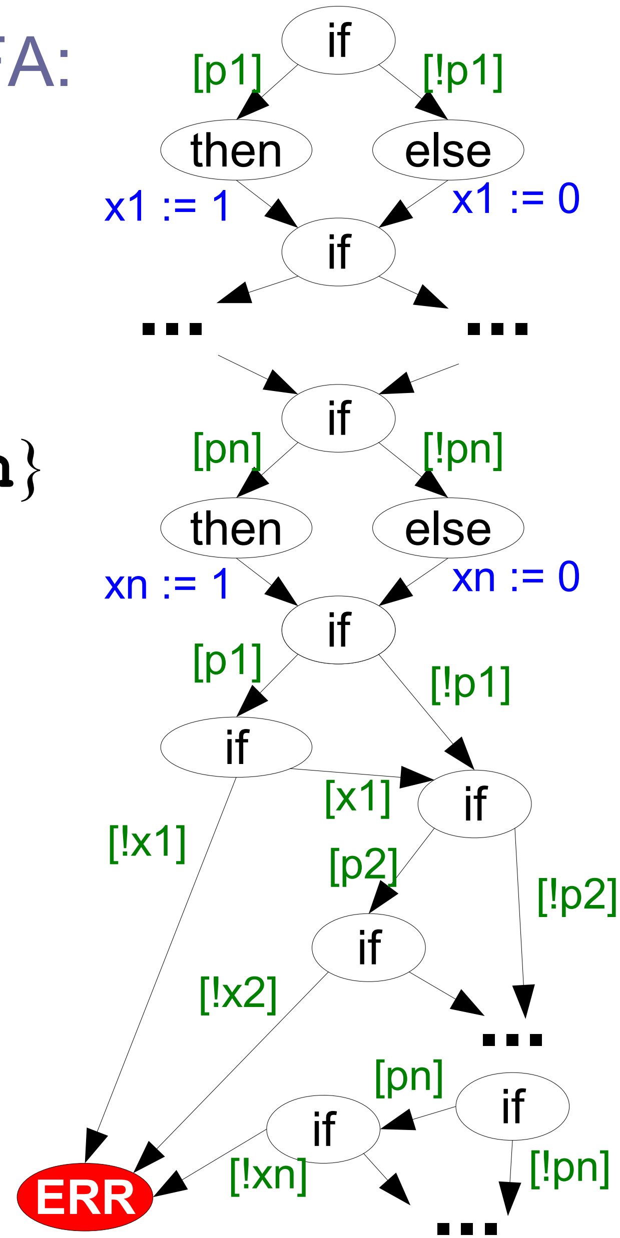


Example of exponential ART

```
if (p1) {x1 = 1;} else {x1 = 0;}
...
if (pn) {xn = 1;} else {xn = 0;}

if (p1) { if (!x1) goto ERROR; }
...
if (pn) { if (!xn) goto ERROR; }
```

Preds needed: { **(x1=0)** ,..., **(xn=0)** , **p1**,...,**pn**}

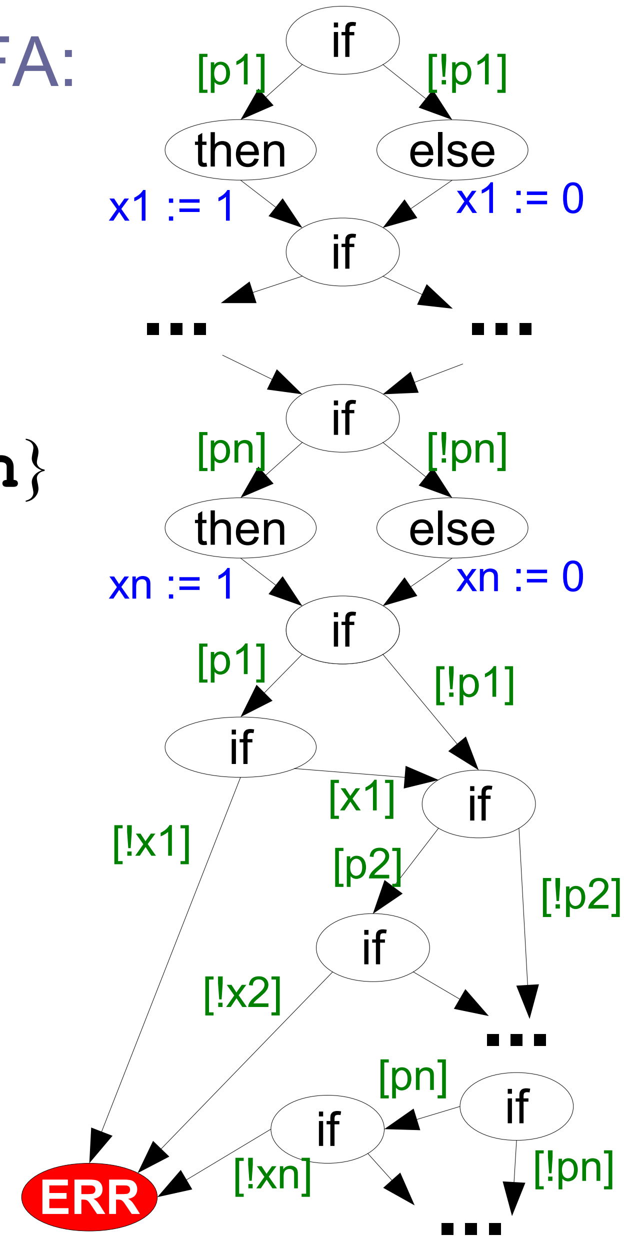
CFA:



Example of exponential ART

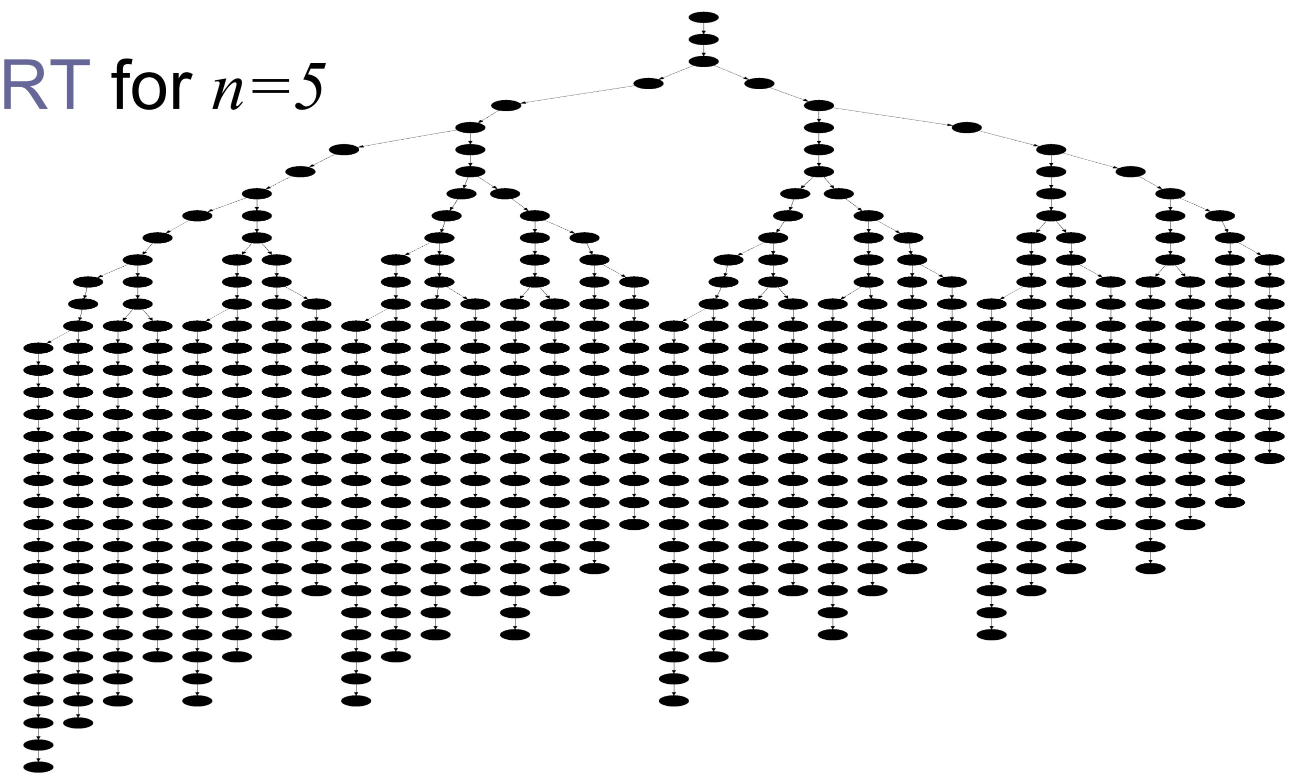
```
if (p1) {x1 = 1;} else {x1 = 0;}
...
if (pn) {xn = 1;} else {xn = 0;}

if (p1) { if (!x1) goto ERROR; }
...
if (pn) { if (!xn) goto ERROR; }
```

CFA:



Preds needed: { **(x1=0)**,..., **(xn=0)**, **p1**,...,**pn**}

ART for *n=5*



Large-Block Encoding

- *Main idea: use a "more coarse-grained" ART*
  - No more 1:1 mapping between ART paths and program paths
  - Each ART edge encodes a loop-free subpart of the program
  - Each ART path encodes a set of program paths

- Consequences:
  - reduce size of the ART (up to exponentially)
  - Increase cost of $\mathsf{SP}_{op}^{\alpha}$, path feasibility checks, refinement

Large-Block Encoding

- *Main idea: use a "more coarse-grained" ART*
  - No more 1:1 mapping between ART paths and program paths
  - Each ART edge encodes a loop-free subpart of the program
  - Each ART path encodes a set of program paths

- Consequences:
  - reduce size of the ART (up to exponentially)
  - Increase cost of $\mathsf{SP}^{\alpha}_{op}$, path feasibility checks, refinement

Less explicit search, more (symbolic) work for the SMT solver

Large-Block Encoding: implementation

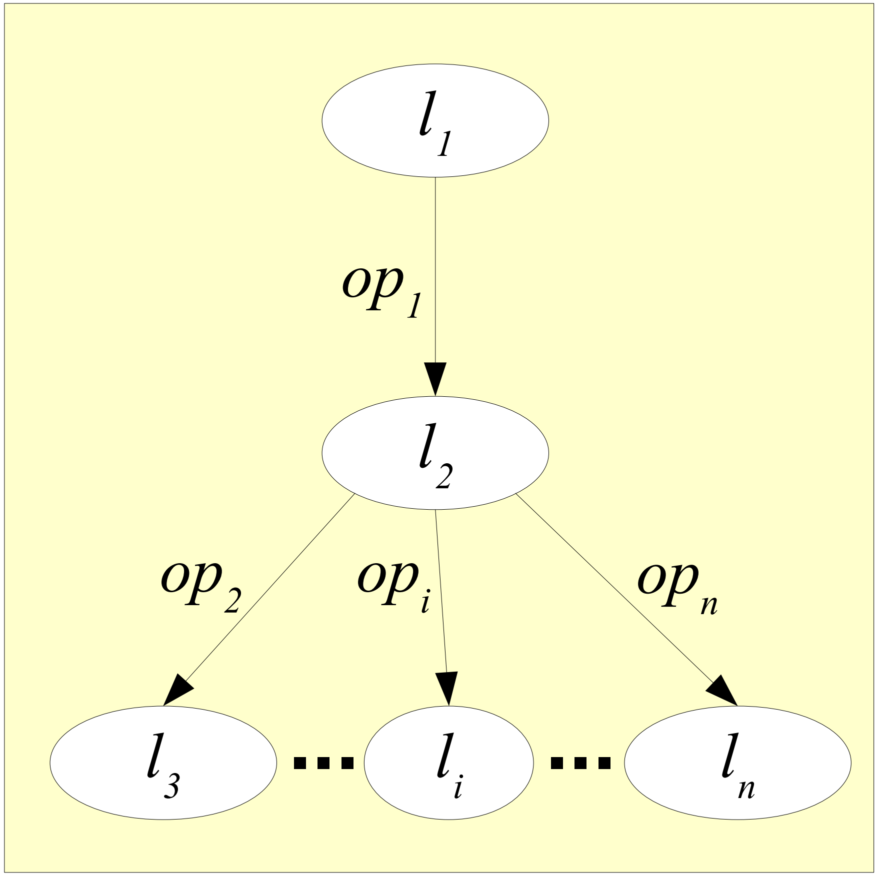
- *Do not modify the algorithm, modify the CFA*
- Summarization of the CFA
  - Fixpoint application of 2 summarization rules

Large-Block Encoding: implementation

- *Do not modify the algorithm, modify the CFA*
- Summarization of the CFA
  - Fixpoint application of 2 summarization rules

Rule 1 (Sequence)

- Conditions:
  - $l_1 \neq l_2$
  - No other incoming edges to l_2

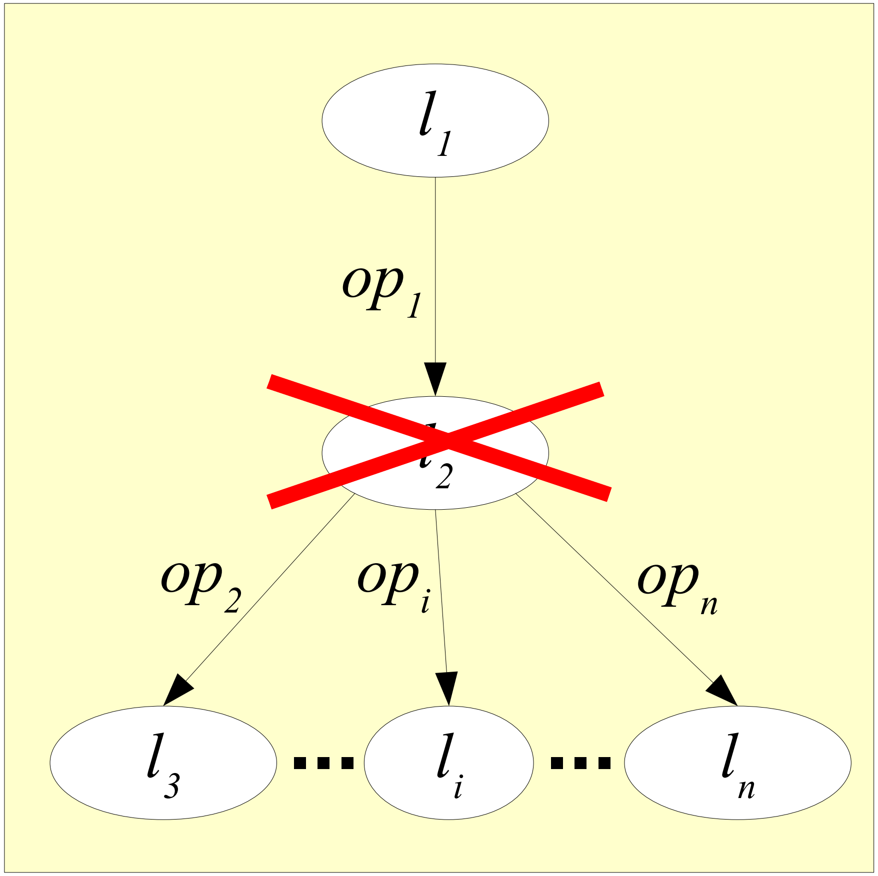


Large-Block Encoding: implementation

- *Do not modify the algorithm, modify the CFA*
- Summarization of the CFA
  - Fixpoint application of 2 summarization rules

Rule 1 (Sequence)

- Conditions:
  - $l_1 \neq l_2$
  - No other incoming edges to l_2



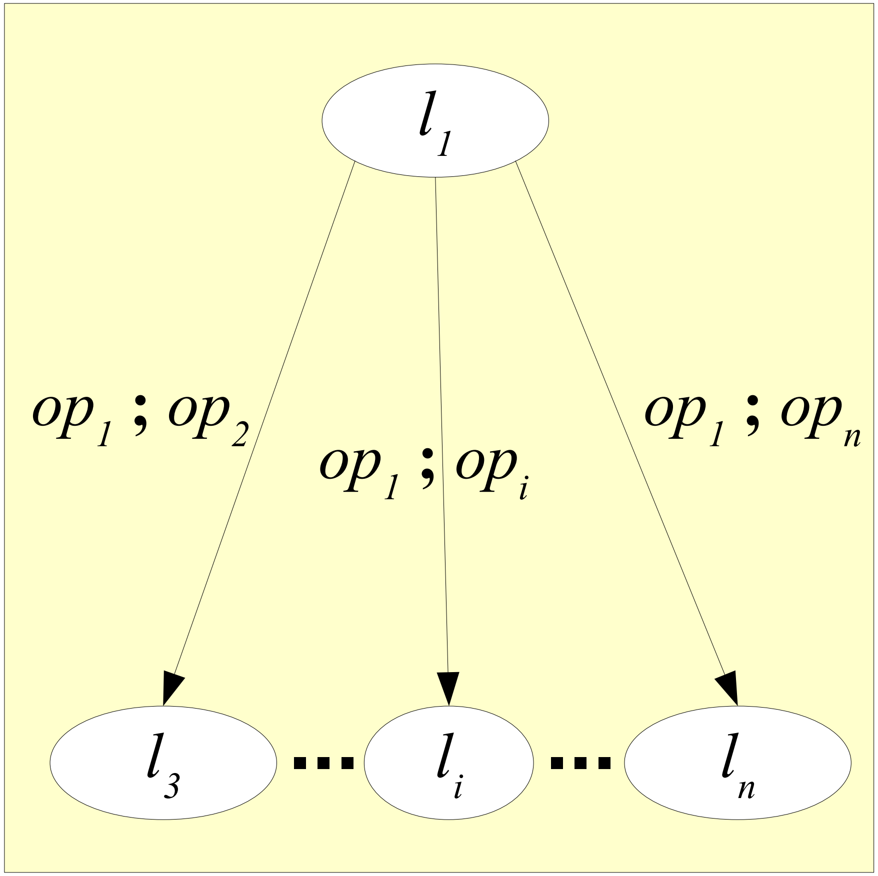
Large-Block Encoding: implementation

- *Do not modify the algorithm, modify the CFA*
- Summarization of the CFA
  - Fixpoint application of 2 summarization rules

Rule 1 (Sequence)

- Conditions:
  - $l_1 \neq l_2$
  - No other incoming edges to l_2

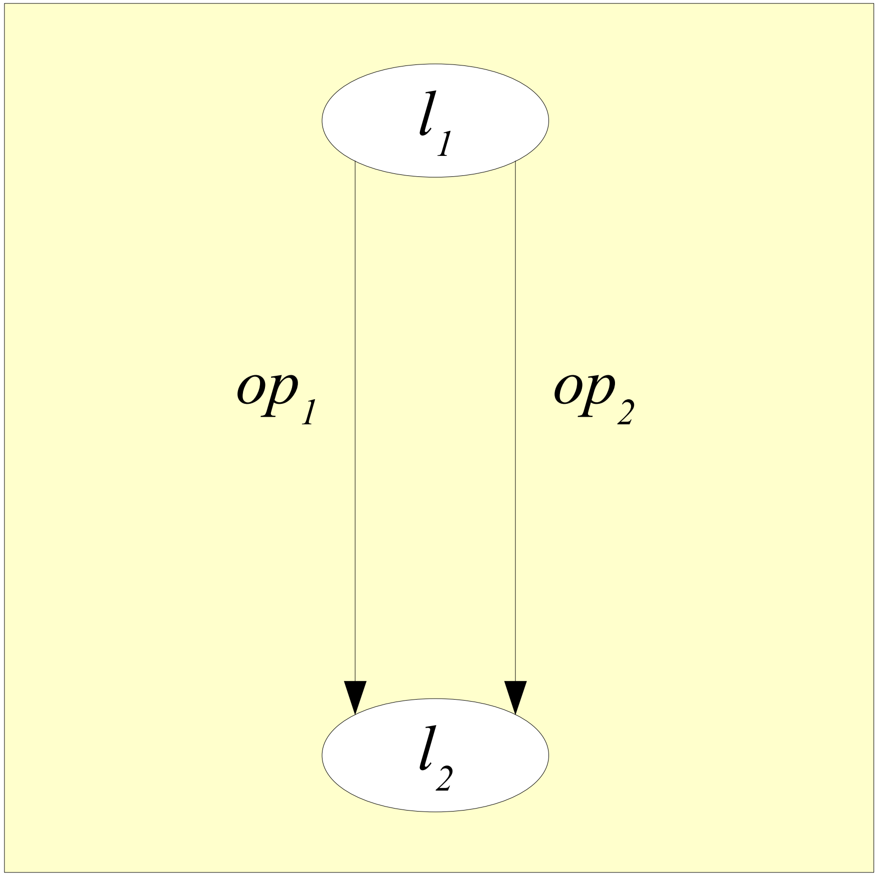
$$\mathsf{SP}_{op_1;op_2}(\varphi) = \mathsf{SP}_{op_2}(\mathsf{SP}_{op_1}(\varphi))$$



Large-Block Encoding: implementation

- *Do not modify the algorithm, modify the CFA*
- Summarization of the CFA
  - Fixpoint application of 2 summarization rules

Rule 2 (Choice)

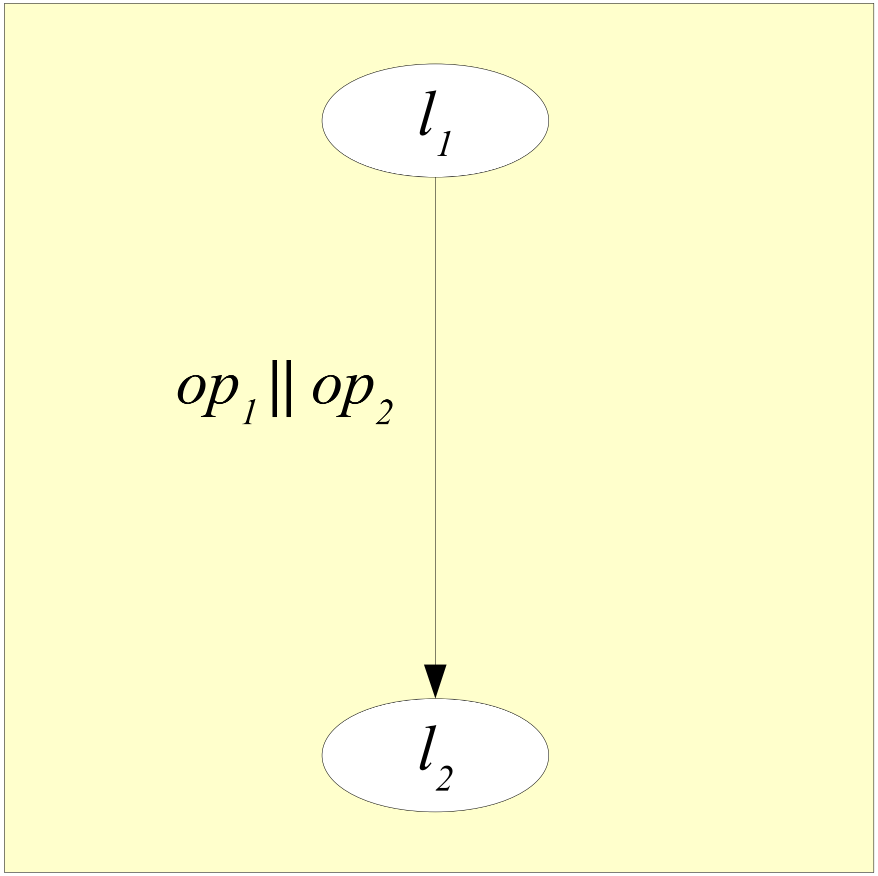


Large-Block Encoding: implementation

- *Do not modify the algorithm, modify the CFA*
- Summarization of the CFA
  - Fixpoint application of 2 summarization rules

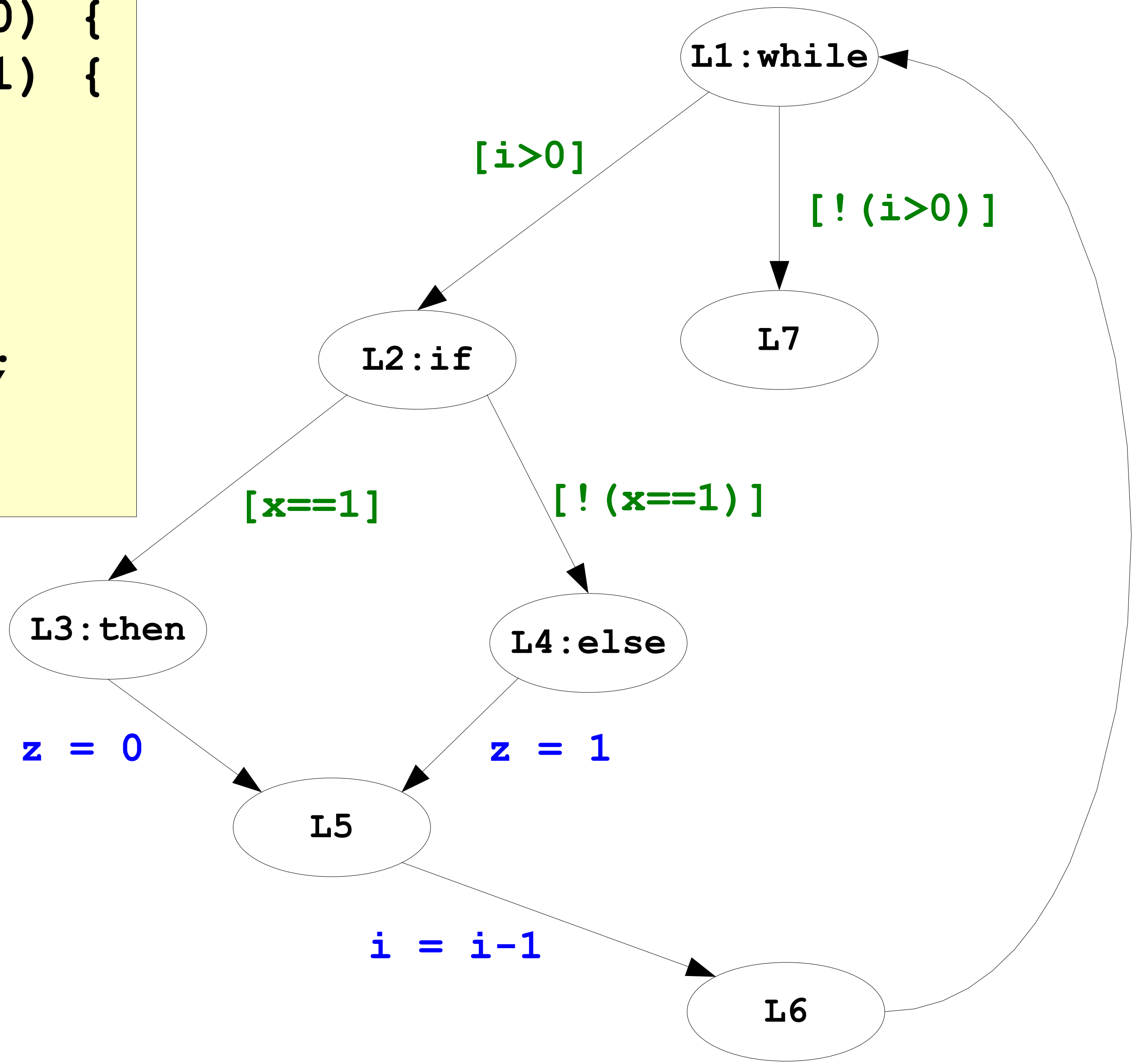
Rule 2 (Choice)

$$\mathsf{SP}_{op_1||op_2}(\varphi) = \mathsf{SP}_{op_1}(\varphi) \vee \mathsf{SP}_{op_2}(\varphi)$$



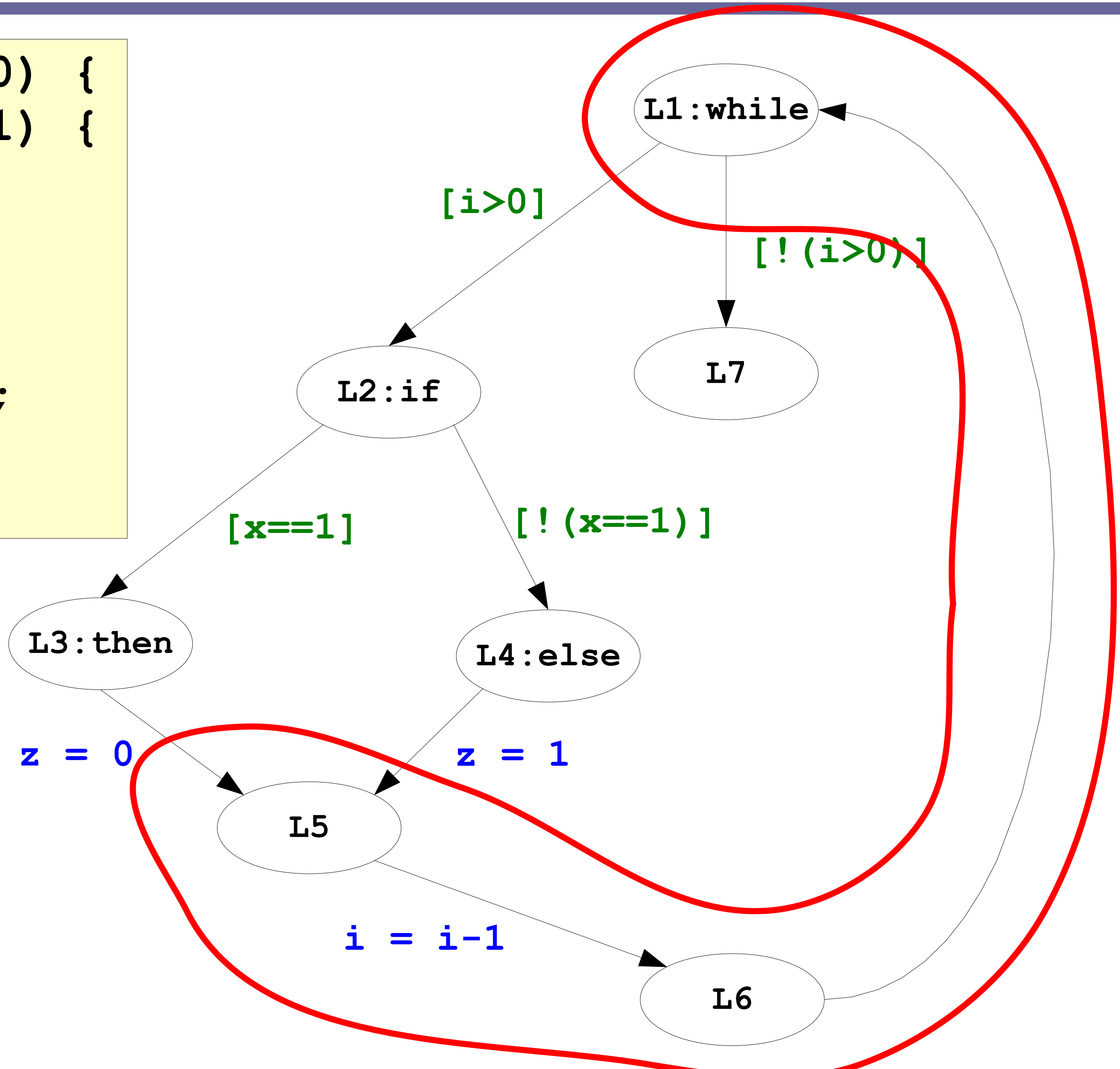
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



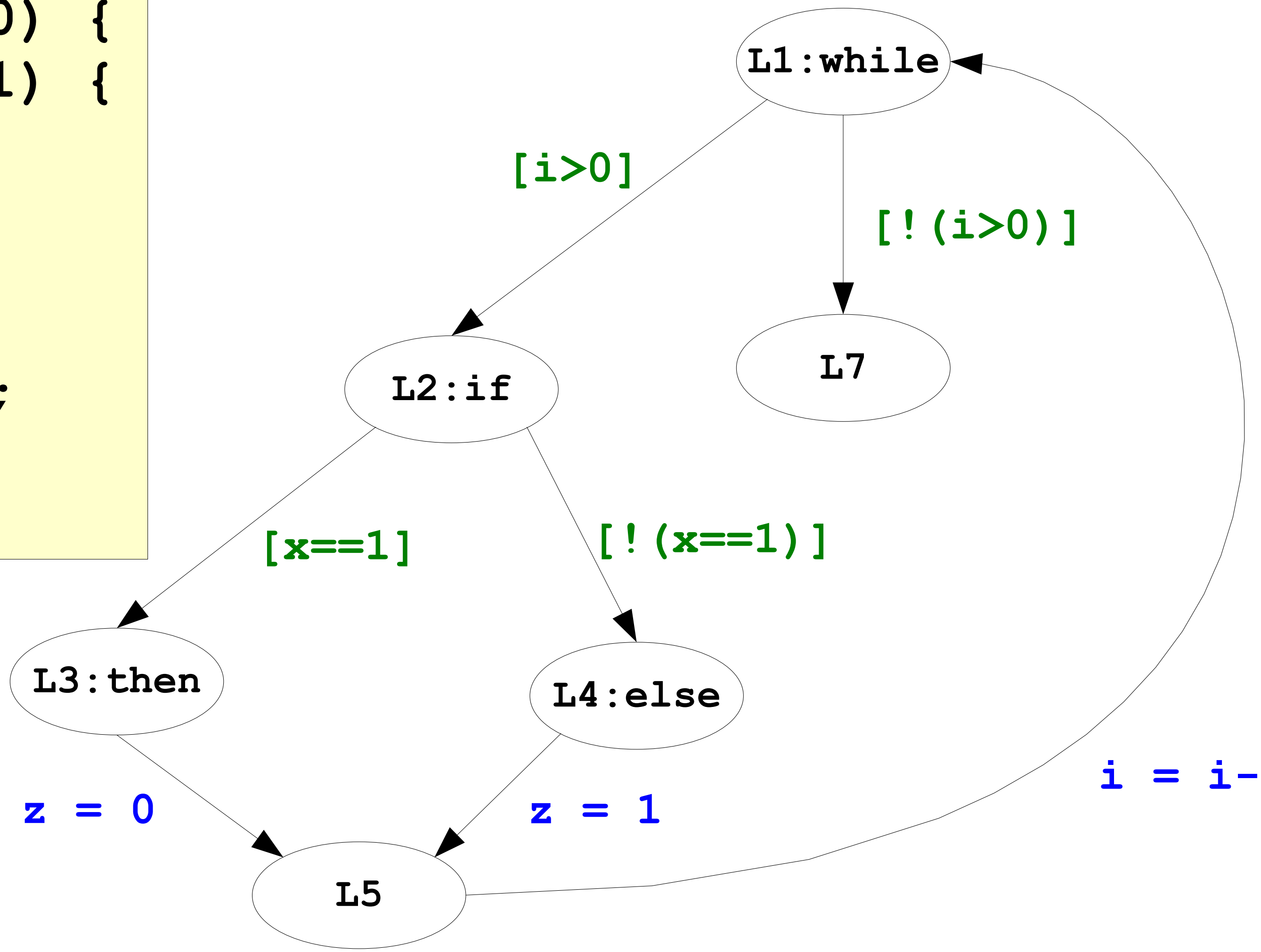
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



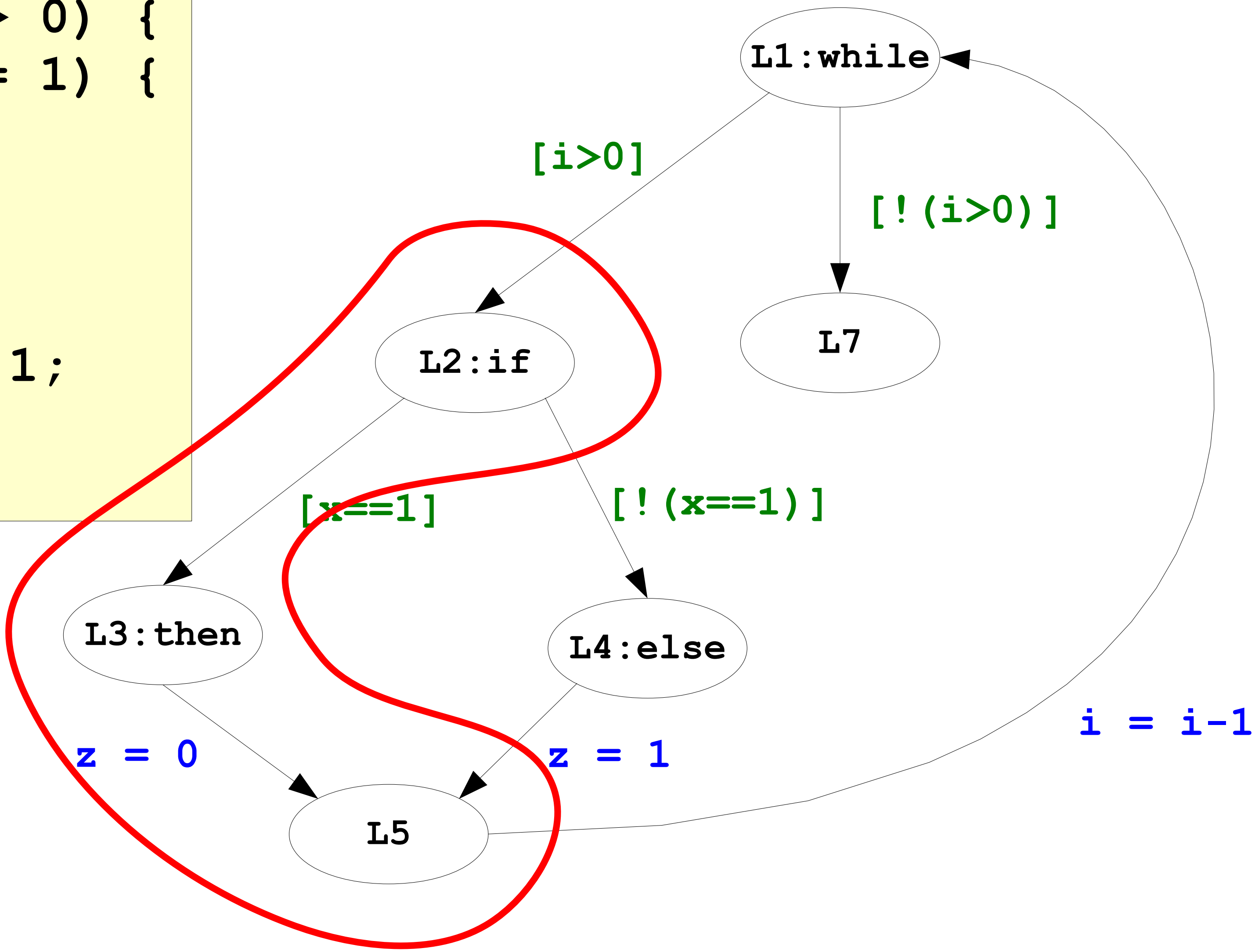
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



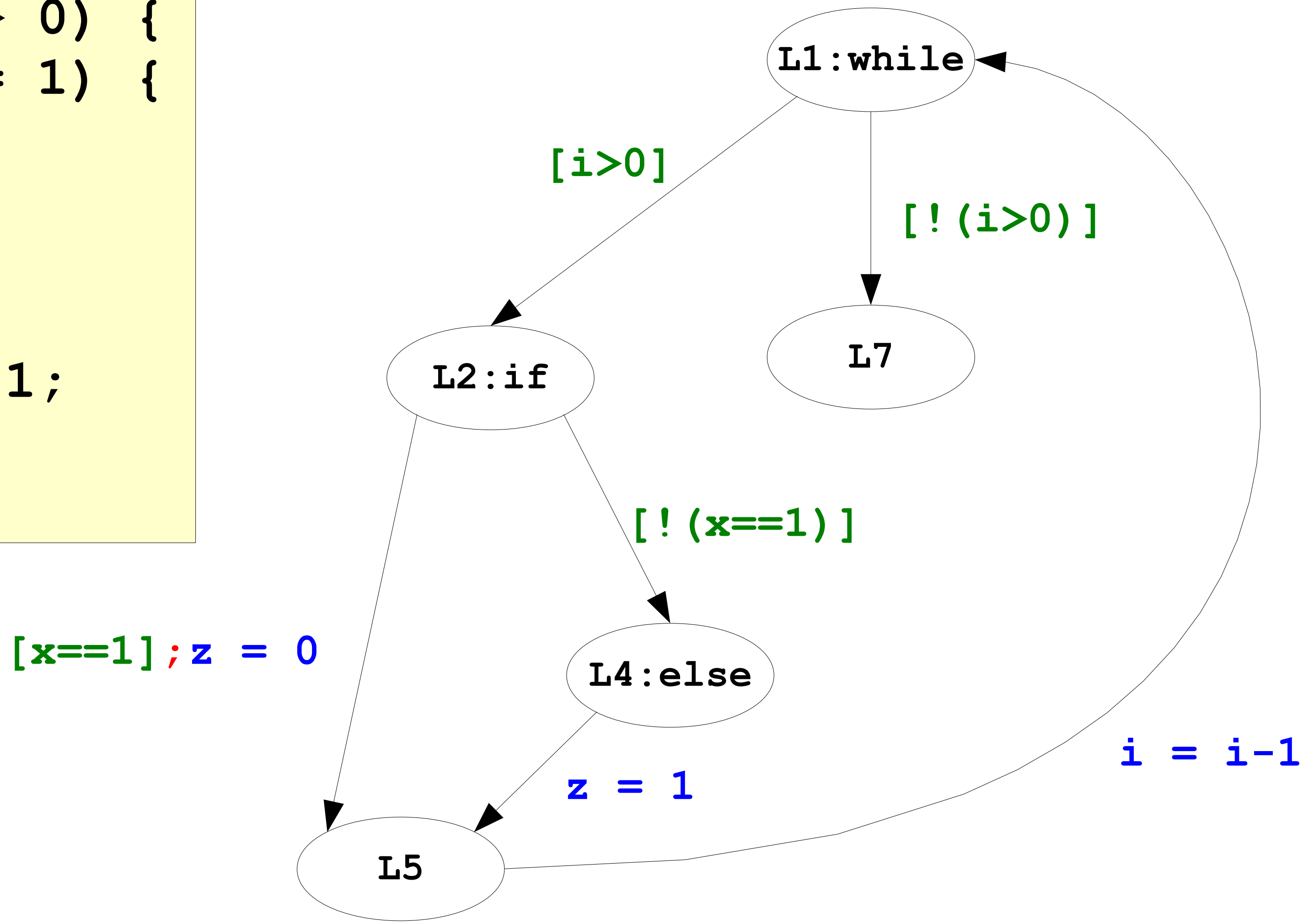
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



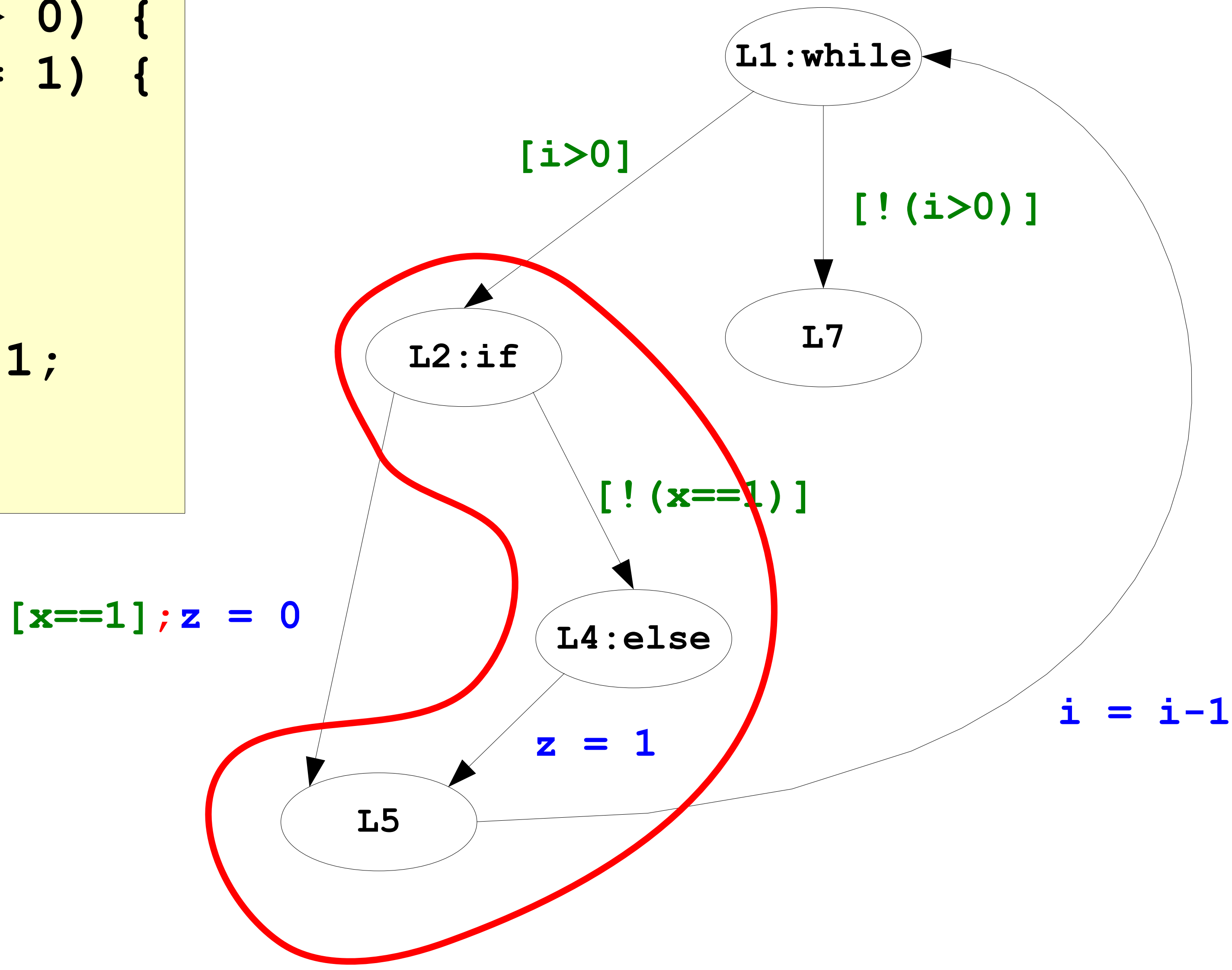
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



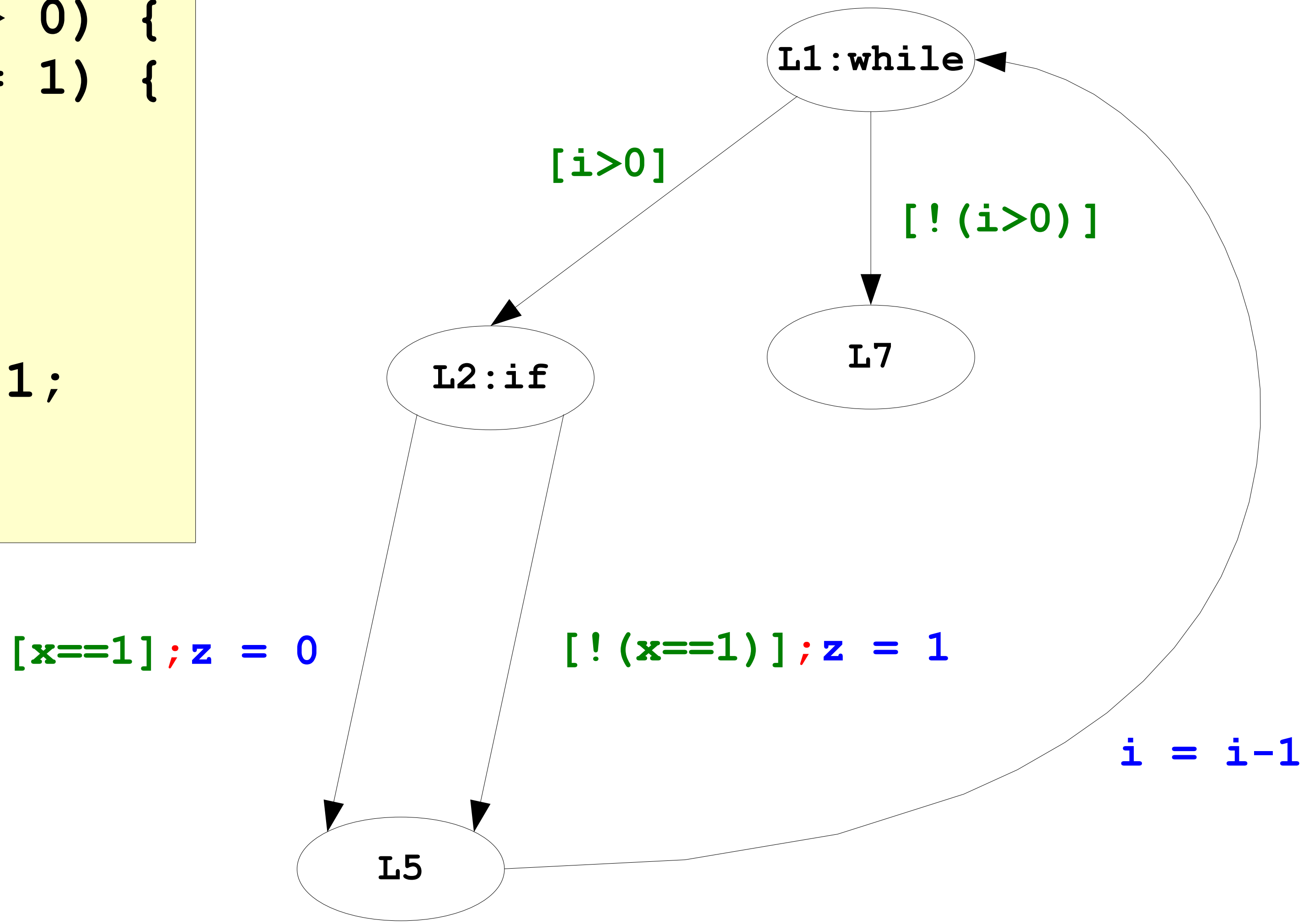
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



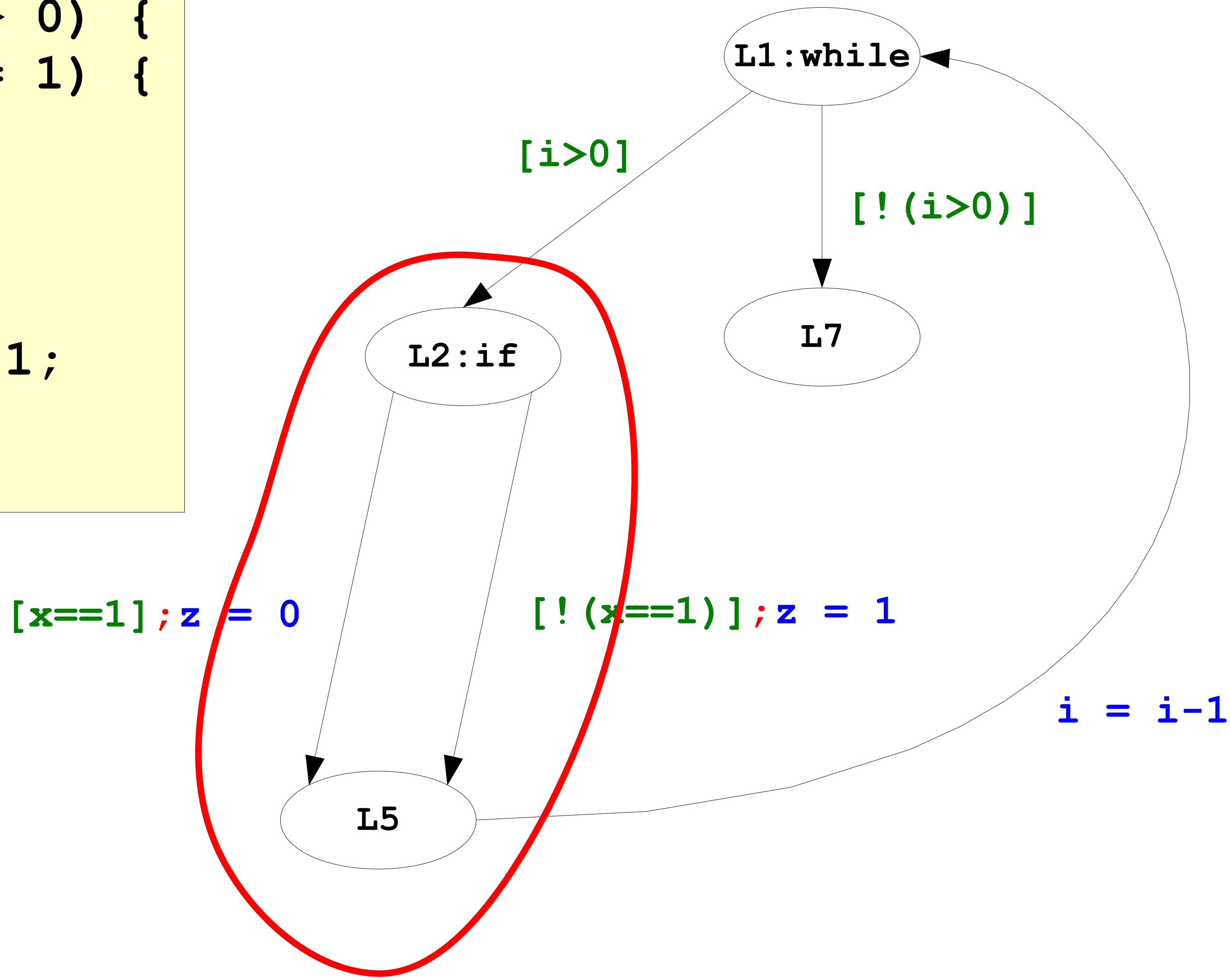
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



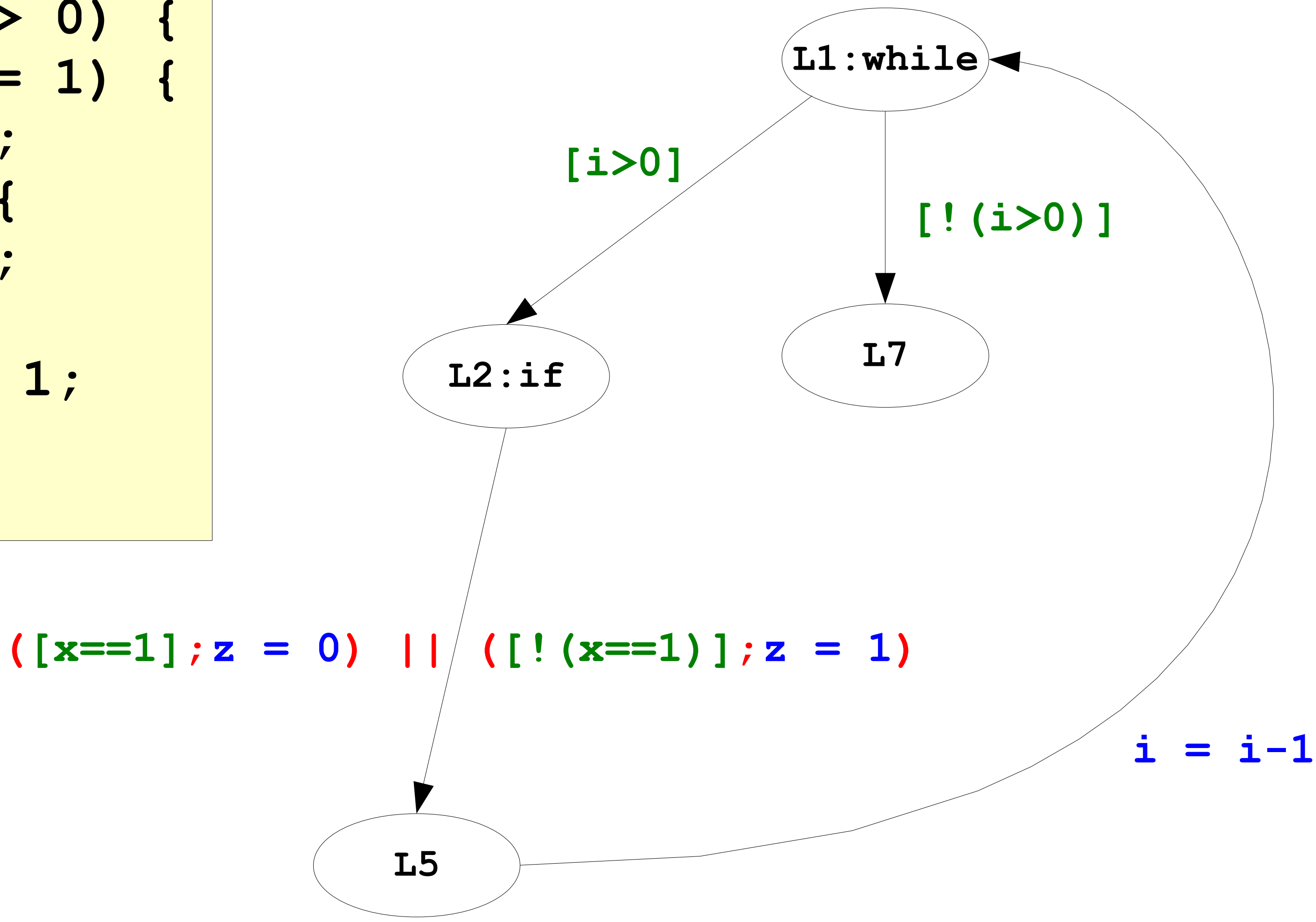
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



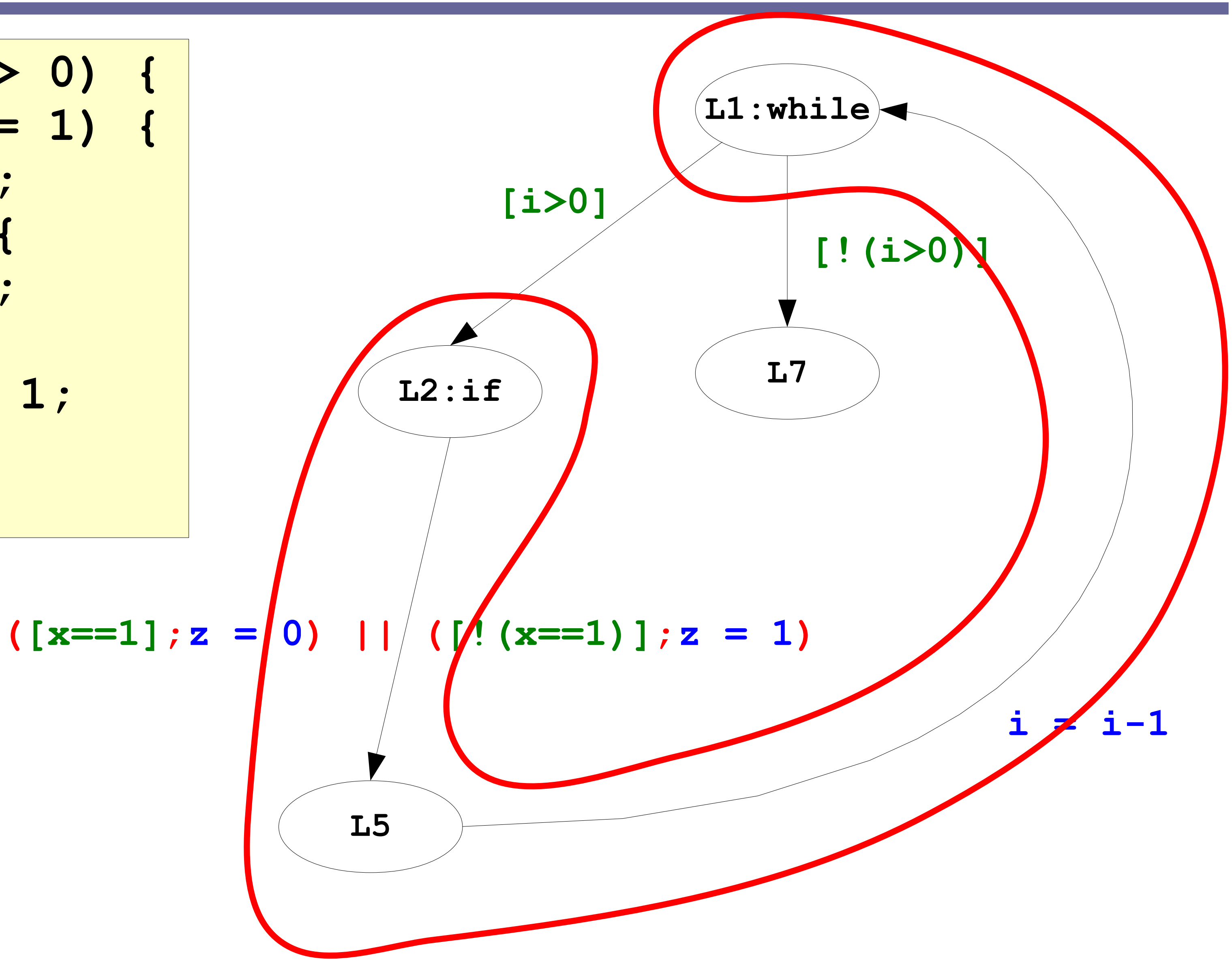
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



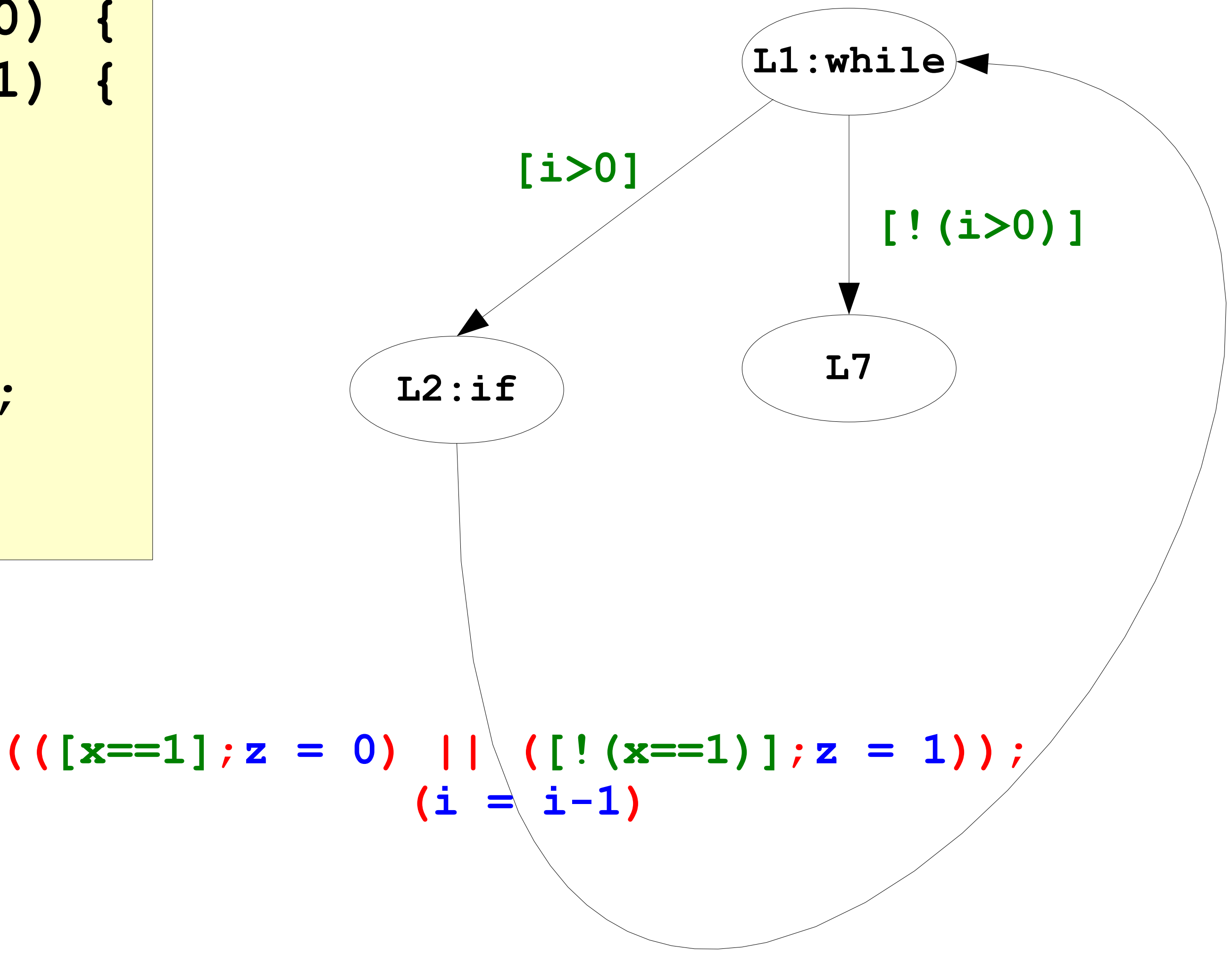
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



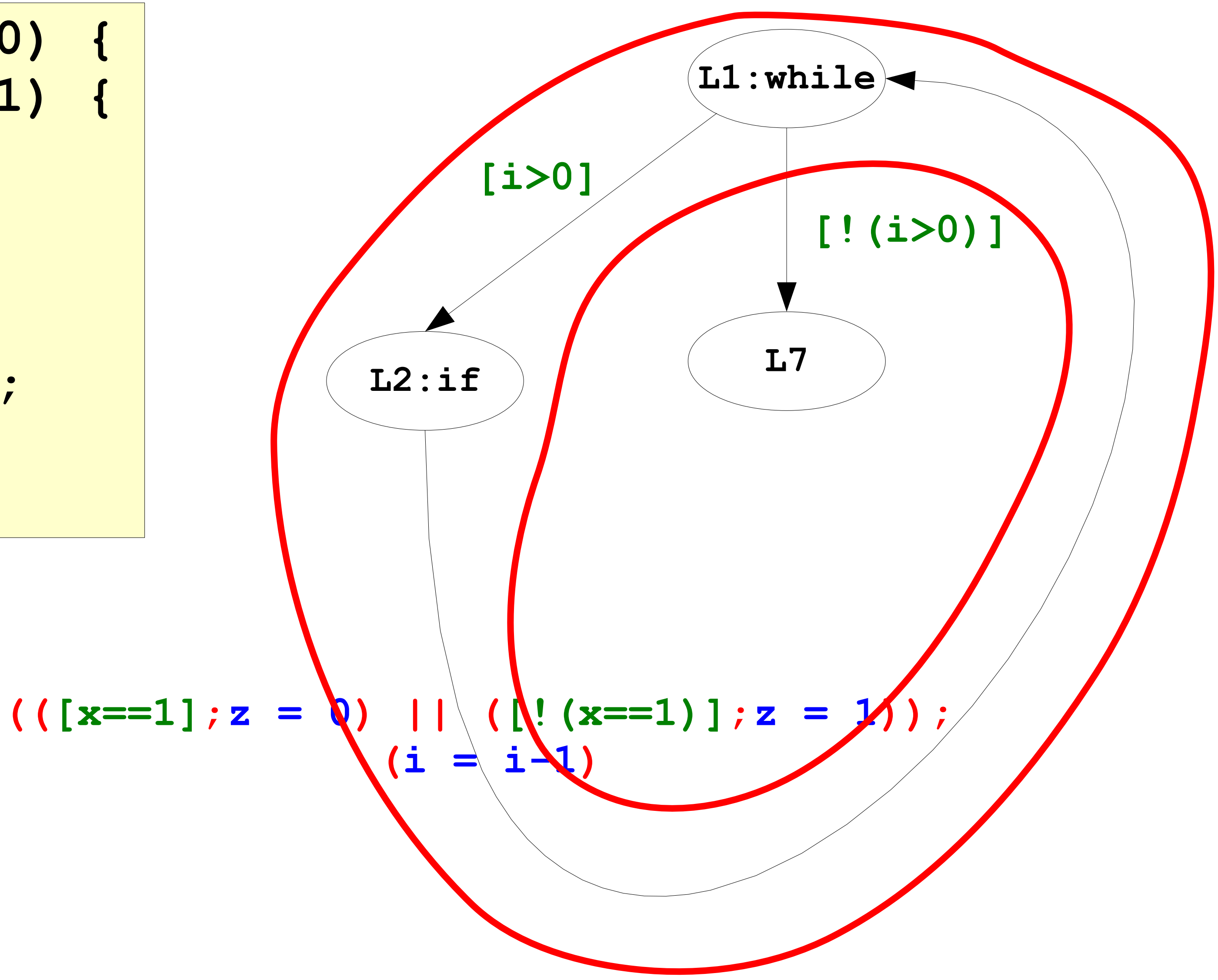
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



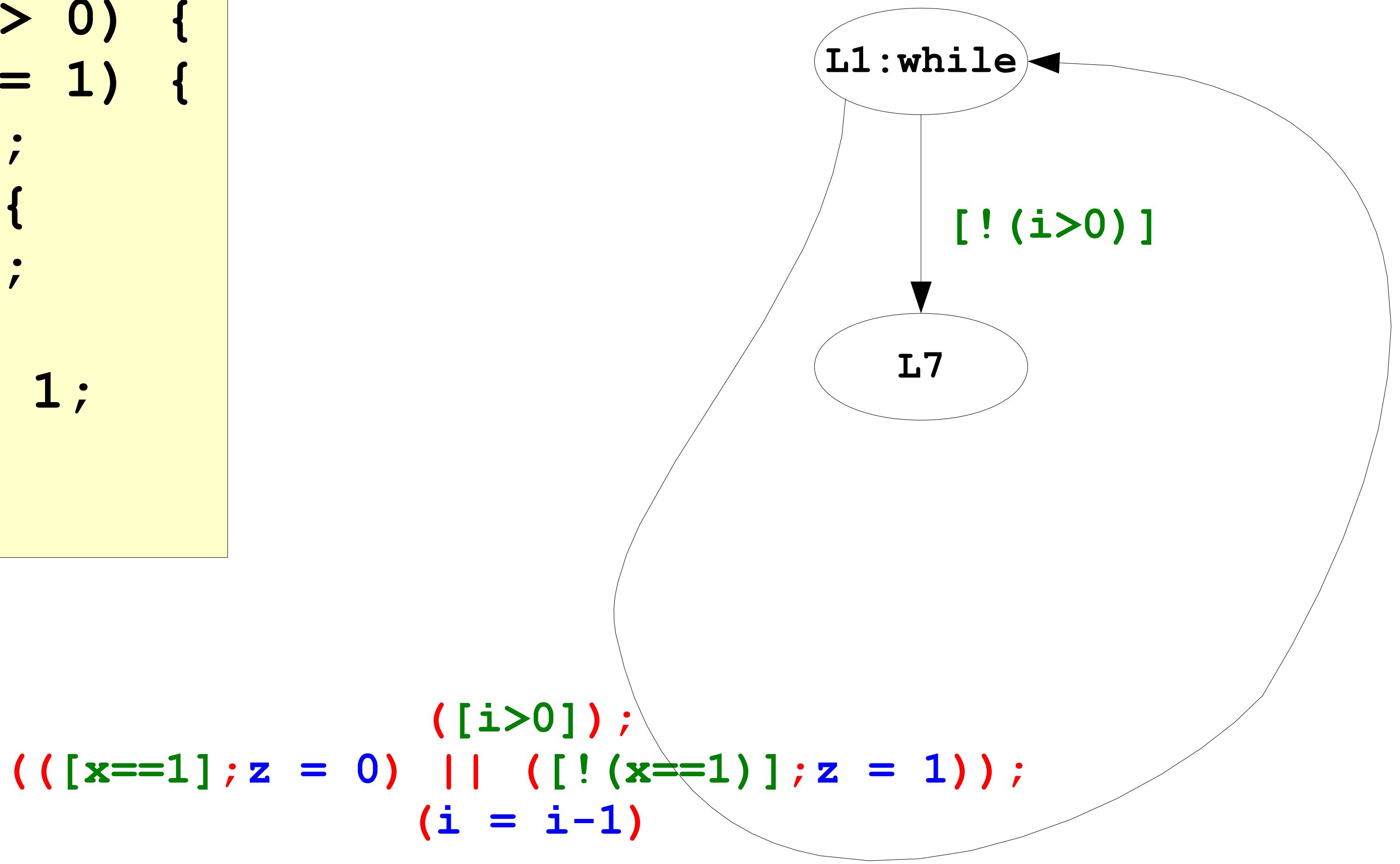
CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



CFA summarization – Example

```
L1: while (i > 0) {
L2:   if (x == 1) {
L3:     z = 0;
      } else {
L4:     z = 1;
      }
L5:   i = i - 1;
L6: }
L7: ...
```



Large blocks and predicate abstraction

- With Large-Block Encoding:
  - Each edge represents a whole loop-free subpart of a program
  - ART paths represent sets of program paths
- Consequence: Cartesian abstraction can't be used anymore
  - Too weak
  - Need to keep track of dependencies among predicates
- In this work: experiment with Boolean abstraction
  - Exponential in the worst case, but can exploit recent work in computation of Boolean abstraction with SMT ([Lahiri et al CAV06], [Cavada et al FMCAD07])
    - Huge progress wrt. previous techniques
- Cost of the analysis moved from explicit path enumeration to symbolic SP^{α} computation

Large blocks and abstraction refinement

- One error trace in the ART corresponds to a set of concrete traces
  - Complexity increased also for counterexample analysis and abstraction-refinement
  - With interpolation-based refinement [Henzinger et al POPL04], exploit improvements in Craig interpolant computation with SMT ([Cimatti et al TACAS08, CADE09], [Fuchs et al TACAS09], [Goel et al CADE09])

Outline

- Background
- Large-Block Encoding
- Experimental evaluation

Experimental evaluation

- Implementation within CPAchecker
  - http://www.cs.sfu.ca/~dbeyer/cpachecker
  - Not only Large-Block Encoding (LBE), but also traditional "Single-Block" (SBE)
- Use MathSAT as SMT backend
  - Boolean abstraction via All-SMT, efficient interpolation for refinement
- Comparison of CPAchecker-LBE vs
  - CPAchecker-SBE
    - Same language, same infrastructure, ...
  - BLAST
    - State-of-the-art ART-based SW MC
    - 4 different configurations tested

Benchmark instances

- test_locks
  - Artificial instances that show exponential blowup of ART with SBE
- ntdrivers
  - Windows NT device drivers
  - “standard” instances used in previous work
  - Taken from BLAST distribution
- SSH
  - Check properties of SSH client and server protocols
  - Taken from BLAST distribution, used in previous work

- For ntdrivers and SSH, test also programs with artificial bugs

Results - test_locks

Program	Blast Best result	Blast -dfs -predH 7	CPAchecker SBE	CPAchecker LBE
test_locks5.c	4.50	4.96	4.01	*0.29*
test_locks6.c	7.81	8.81	7.22	*0.34*
test_locks7.c	13.91	15.15	12.63	*0.34*
test_locks8.c	25.00	26.49	23.93	*0.57*
test_locks9.c	46.84	49.29	52.04	*0.38*
test_locks10.c	94.57	97.85	131.39	*0.40*
test_locks11.c	204.55	208.78	MO	*0.70*
test_locks12.c	529.16	533.97	MO	*0.46*
test_locks13.c	1229.27	1232.87	MO	*0.49*
test_locks14.c	>1800.00	>1800.00	MO	*0.50*
test_locks15.c	>1800.00	>1800.00	MO	*0.29*
TOTAL	**9 / 2155.61**	**9 / 2178.17**	**6 / 231.22**	**11 / 4.76**

Results - test_locks

	Blast		CPAchecker	
Program	**Best result**	**-dfs -predH 7**	**SBE**	**LBE**
test_locks5.c	4.50	4.96	4.01	***0.29***
test_locks6.c	7.81	8.81	7.22	***0.34***
test_locks7.c	13.91	15.15	12.63	***0.34***
test_locks8.c	25.00	26.49	23.93	***0.57***
test_loc[illegible]	[illegible]	[illegible]9.29	52.04	***0.38***
test_loc[illegible]	[illegible]	[illegible]7.85	131.39	***0.40***
test_loc[illegible]	[illegible]	[illegible]8.78	MO	***0.70***
test_locks12.c	529.16	533.97	MO	***0.46***
test_locks13.c	1229.27	1232.87	MO	***0.49***
test_locks14.c	>1800.00	>1800.00	MO	***0.50***
test_locks15.c	>1800.00	>1800.00	MO	***0.29***
TOTAL	**9 / 2155.61**	**9 / 2178.17**	**6 / 231.22**	**11 / 4.76**

2 more instances,
>500x faster

Results – ntdrivers and SSH – safe

	Blast		CPAchecker	
Program	Best result	-dfs -predH 7	SBE	LBE
cdaudio	175.76	264.12	MO	*53.55*
diskperf	>1800.00	>1800.00	MO	*232.00*
floppy	218.26	>1800.00	MO	*56.36*
kbfiltr	23.55	32.80	41.12	*7.82*
parport	738.82	915.79	MO	*378.04*
s3_clnt.01	33.01	1000.41	755.81	*19.51*
s3_clnt.02	62.65	312.77	1075.45	*16.00*
s3_clnt.03	60.62	314.74	746.31	*49.50*
s3_clnt.04	63.96	197.65	730.80	*25.45*
s3_srvr.01	811.27	1036.89	>1800.00	*125.33*
s3_srvr.02	360.47	360.47	>1800.00	*122.83*
s3_srvr.03	276.19	276.19	>1800.00	*98.47*
s3_srvr.04	175.64	301.85	>1800.00	*71.77*
s3_srvr.06	304.63	304.63	>1800.00	*59.70*
s3_srvr.07	478.05	666.53	>1800.00	*85.82*
s3_srvr.08	115.76	115.76	>1800.00	*61.29*
s3_srvr.09	445.21	1037.09	>1800.00	*126.47*
s3_srvr.10	115.10	115.10	>1800.00	*63.36*
s3_srvr.11	367.98	844.28	>1800.00	*162.76*
s3_srvr.12	304.05	304.05	>1800.00	*170.33*
s3_srvr.13	580.33	878.54	>1800.00	*74.49*
s3_srvr.14	303.21	303.21	>1800.00	*50.38*
s3_srvr.15	115.88	115.88	>1800.00	*21.01*
s3_srvr.16	305.11	305.11	>1800.00	*127.82*
TOTAL	**23 / 6435.51**	**22 / 10003.06**	**5 / 3349.48**	**24 / 2260.07**

Results – ntdrivers and SSH – safe

	Blast		CPAchecker	
Program	**Best result**	**-dfs -predH 7**	**SBE**	**LBE**
cdaudio	175.76	264.12	MO	*53.55*
diskperf	>1800.00	>1800.00	MO	*232.00*
floppy	218.26	>1800.00	MO	*56.36*
kbfiltr	23.55	32.80	41.12	*7.82*
parport	738.82	915.79	MO	*378.04*
s3_clnt.01	33.01	1000.41	755.81	*19.51*
s3_clnt.02	62.65	312.77	1075.45	*16.00*
s3_clnt.03	60.62	314.74	746.31	*49.50*
s3_clnt.04	63.96	197.65	730.80	*25.45*
s3_srvr.01	811.27	1036.89	>1800.00	*125.33*
s3_srvr.02	360.47	360.47	>1800.00	*122.83*
s3_srvr.03	276.19	276.19	>1800.00	*98.47*
s3_srvr.04	175.64	301.85	>1800.00	*71.77*
s3_srvr.06	[illegible]	[illegible]	>1800.00	*59.70*
s3_srvr.07	[illegible]	[illegible]	>1800.00	*85.82*
s3_srvr.08	[illegible]	[illegible]	>1800.00	*61.29*
s3_srvr.09	[illegible]	[illegible]	>1800.00	*126.47*
s3_srvr.10	115.10	115.10	>1800.00	*63.36*
s3_srvr.11	367.98	344.28	>1800.00	*162.76*
s3_srvr.12	304.05	304.05	>1800.00	*170.33*
s3_srvr.13	580.33	878.54	>1800.00	*74.49*
s3_srvr.14	303.21	303.21	>1800.00	*50.38*
s3_srvr.15	115.88	115.88	>1800.00	*21.01*
s3_srvr.16	305.11	305.11	>1800.00	*127.82*
TOTAL	**23 / 6435.51**	**22 / 10003.06**	**5 / 3349.48**	**24 / 2260.07**

1 more instance,
~3.2x faster

Results – ntdrivers and SSH – safe

Program	Blast: Best result	Blast: -dfs -predH 7	CPAchecker: SBE	CPAchecker: LBE
cdaudio	175.76	264.12	MO	***53.55***
diskperf	>1800.00	>1800.00	MO	***232.00***
floppy	218.26	>1800.00	MO	***56.36***
kbfiltr	23.55	32.80	41.12	***7.82***
parport	738.82	915.79	MO	***378.04***
s3_clnt.01	33.01	1000.41	755.81	***19.51***
s3_clnt.02	62.65	312.77	1075.45	***16.00***
s3_clnt.03	60.62	314.74	746.31	***49.50***
s3_clnt.04	63.96	197.65	730.80	***25.45***
s3_srvr.01	811.27	1036.89	>1800.00	***125.33***
s3_srvr.02	360.47	360.47	>1800.00	***122.83***
s3_srvr.03	276.19	276.19	>1800.00	***98.47***
s3_srvr.04	175.64	301.85	>1800.00	***71.77***
s3_srvr.06	[illegible]	[illegible]4.63	>1800.00	***59.70***
s3_srvr.07	[illegible]	[illegible]6.53	>1800.00	***85.82***
s3_srvr.08	[illegible]	[illegible]5.76	>1800.00	***61.29***
s3_srvr.09	[illegible]	[illegible]7.09	>1800.00	***126.47***
s3_srvr.10	[illegible]	115.10	>1800.00	***63.36***
s3_srvr.11	367.98	344.28	>1800.00	***162.76***
s3_srvr.12	304.05	304.05	>1800.00	***170.33***
s3_srvr.13	580.33	878.54	>1800.00	***74.49***
s3_srvr.14	303.21	303.21	>1800.00	***50.38***
s3_srvr.15	115.88	115.88	>1800.00	***21.01***
s3_srvr.16	305.11	305.11	>1800.00	***127.82***
TOTAL	**23 / 6435.51**	**22 / 10003.06**	**5 / 3349.4[illegible]**	**24 / 2260.07**

2 more instances,
~5x faster

Results – ntdrivers and SSH – unsafe

Program	Blast		CPAchecker	
	Best result	-bfs -predH 7	SBE	LBE
cdaudio	18.79	99.82	74.39	***9.85***
diskperf	889.79	>1800.00	26.53	***6.78***
floppy	119.60	>1800.00	36.49	***4.30***
kbfiltr	46.80	144.25	75.45	***11.52***
parport	***1.67***	10.95	14.62	2.64
s3_clnt.01	8.84	28.30	1514.90	***3.33***
s3_clnt.02	9.02	9.02	843.42	***3.27***
s3_clnt.03	6.64	6.64	780.72	***2.61***
s3_clnt.04	9.78	9.78	724.04	***3.18***
s3_srvr.01	7.59	7.59	MO	***2.09***
s3_srvr.02	7.16	7.16	>1800.00	***2.10***
s3_srvr.03	7.42	7.42	>1800.00	***2.08***
s3_srvr.04	7.33	7.33	>1800.00	***1.93***
s3_srvr.06	39.81	56.11	MO	***5.08***
s3_srvr.07	310.84	310.84	>1800.00	***28.35***
s3_srvr.08	40.51	73.59	>1800.00	***36.47***
s3_srvr.09	265.48	265.48	>1800.00	***4.94***
s3_srvr.10	40.24	66.88	>1800.00	***12.01***
s3_srvr.11	49.05	49.05	>1800.00	***4.80***
s3_srvr.12	38.66	38.66	>1800.00	***6.11***
s3_srvr.13	251.56	251.56	>1800.00	***15.20***
s3_srvr.14	39.94	53.93	1656.54	***4.63***
s3_srvr.15	40.19	77.51	>1800.00	***10.19***
s3_srvr.16	39.54	55.97	>1800.00	***5.21***
TOTAL	**24 / 2296.25**	**22 / 1637.84**	**10 / 5747.10**	**24 / 188.67**

Results – ntdrivers and SSH – unsafe

Program	Blast: Best result	Blast: -bfs -predH 7	CPAchecker: SBE	CPAchecker: LBE
cdaudio	18.79	99.82	74.39	*9.85*
diskperf	889.79	>1800.00	26.53	*6.78*
floppy	119.60	>1800.00	36.49	*4.30*
kbfiltr	46.80	144.25	75.45	*11.52*
parport	*1.67*	10.95	14.62	2.64
s3_clnt.01	8.84	28.30	1514.90	*3.33*
s3_clnt.02	9.02	9.02	843.42	*3.27*
s3_clnt.03	6.64	6.64	780.72	*2.61*
s3_clnt.04	9.78	9.78	724.04	*3.18*
s3_srvr.01	7.59	7.59	MO	*2.09*
s3_srvr.02	7.16	7.16	>1800.00	*2.10*
s3_srvr.03	7.42	7.42	>1800.00	*2.08*
s3_srvr.04	7.33	7.33	>1800.00	*1.93*
s3_srvr.06	[illegible]	6.11	MO	*5.08*
s3_srvr.07	[illegible]	[illegible]	>1800.00	*28.35*
s3_srvr.08	[illegible]	[illegible]	>1800.00	*36.47*
s3_srvr.09	[illegible]	[illegible]	>1800.00	*4.94*
s3_srvr.10	[illegible]	[illegible]	>1800.00	*12.01*
s3_srvr.11	49.05	49.05	>1800.00	*4.80*
s3_srvr.12	38.66	38.66	>1800.00	*6.11*
s3_srvr.13	251.56	251.56	>1800.00	*15.20*
s3_srvr.14	39.94	53.93	1656.54	*4.63*
s3_srvr.15	40.19	77.51	>1800.00	*10.19*
s3_srvr.16	39.54	55.97	>1800.00	*5.21*
TOTAL	**24 / 2296.25**	**22 / 1637.84**	**10 / 5747.1[illegible]**	**24 / 188.67**

~12x faster

Results – ntdrivers and SSH – unsafe

Program	Blast: Best result	Blast: -bfs -predH 7	CPAchecker: SBE	CPAchecker: LBE
cdaudio	18.79	99.82	74.39	***9.85***
diskperf	889.79	>1800.00	26.53	***6.78***
floppy	119.60	>1800.00	36.49	***4.30***
kbfiltr	46.80	144.25	75.45	***11.52***
parport	***1.67***	10.95	14.62	2.64
s3_clnt.01	8.84	28.30	1514.90	***3.33***
s3_clnt.02	9.02	9.02	843.42	***3.27***
s3_clnt.03	6.64	6.64	780.72	***2.61***
s3_clnt.04	9.78	9.78	724.04	***3.18***
s3_srvr.01	7.59	7.59	MO	***2.09***
s3_srvr.02	7.16	7.16	>1800.00	***2.10***
s3_srvr.03	7.42	7.42	>1800.00	***2.08***
s3_srvr.04	7.33	7.33	>1800.00	***1.93***
s3_srvr.06	[illegible]	6.11	MO	***5.08***
s3_srvr.07	[illegible]	[illegible]0.84	>1800.00	***28.35***
s3_srvr.08	[illegible]	[illegible]3.59	>1800.00	***36.47***
s3_srvr.09	[illegible]	[illegible]5.48	>1800.00	***4.94***
s3_srvr.10	40.24	[illegible]6.88	>1800.00	***12.01***
s3_srvr.11	49.05	49.05	>1800.00	***4.80***
s3_srvr.12	38.66	38.66	>1800.00	***6.11***
s3_srvr.13	251.56	251.56	>1800.00	***15.20***
s3_srvr.14	39.[illegible]4	53.93	1656.54	***4.63***
s3_srvr.15	40.19	77.51	>1800.00	***10.19***
s3_srvr.16	39.54	55.97	>1800.00	***5.21***
TOTAL	**24 / 2296.25**	**22 / 1637.84**	**10 / 5747.1[illegible]**	**24 / 188.67**

2 more instances,
~9.2x faster

Conclusions

- Large-Block Encoding: new approach to ART-based SW MC, for better exploiting modern SMT solvers
  - Move cost from explicit path enumeration to symbolic computations within the SMT solver
  - Nice improvements on standard benchmark programs

- Future work
  - Dynamic computation of large blocks
    - Allows for adjusting the symbolic/explicit tradeoff “on the fly”
  - Experiment with approximated abstractions, cheaper than Boolean
  - Extend to McMillan's CAV'06 approach (no predicate abstraction, use interpolants directly)