Computer aided verification

Lecture 8:

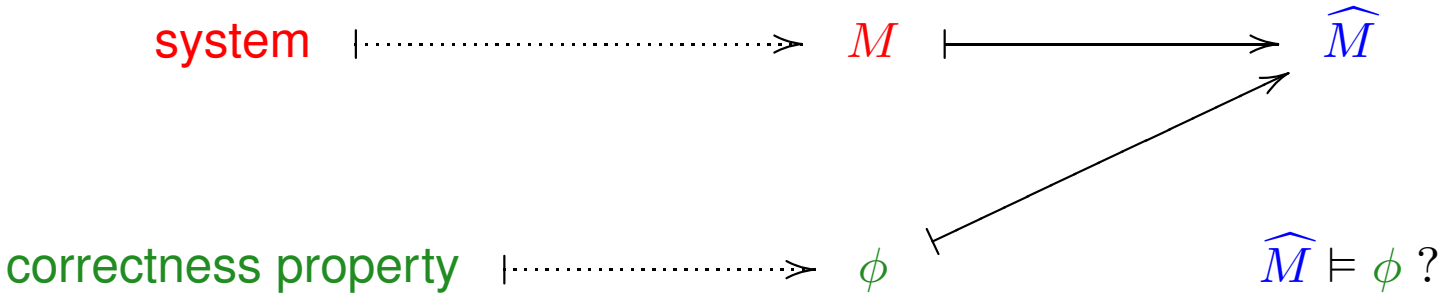
Abstraction

Sławomir Lasota
University of Warsaw

Abstraction

– Model $M = (S, S_0, L, R)$

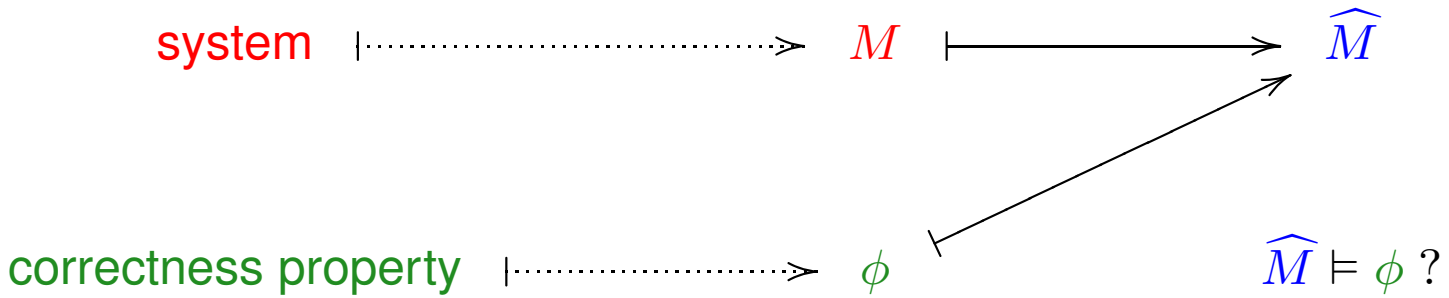
– Abstraction = reduction of the size of the model by removing irrelevant details



Abstraction

– Model $M = (S, S_0, L, R)$

– Abstraction = reduction of the size of the model by removing irrelevant details

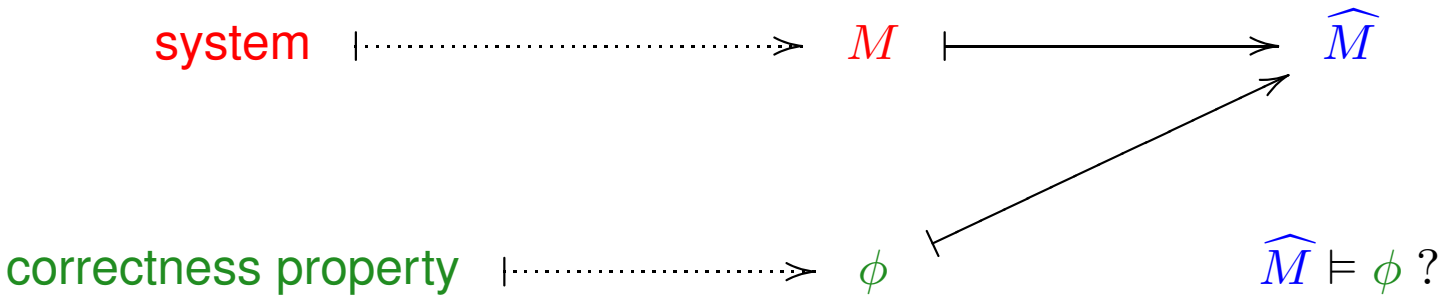


– We hope that computing $\widehat{M}$ and checking $\widehat{M} \models \phi$ is cheaper than checking $M \models \phi$.

Abstraction

– Model $M = (S, S_0, L, R)$

– Abstraction = reduction of the size of the model by removing irrelevant details

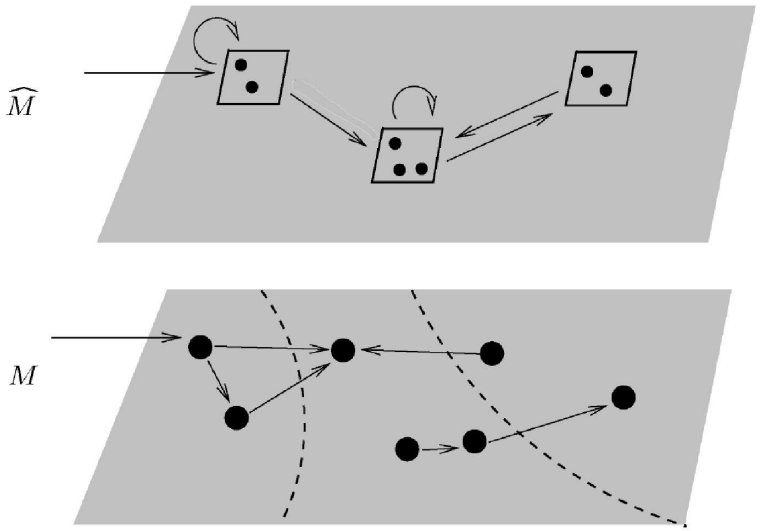


– We hope that computing $\widehat{M}$ and checking $\widehat{M} \models \phi$ is cheaper than checking $M \models \phi$.

– Abstract model $\widehat{M} = (\widehat{S}, \widehat{S_0}, \widehat{L}, \widehat{R})$ (concrete model $M = (S, S_0, L, R)$)

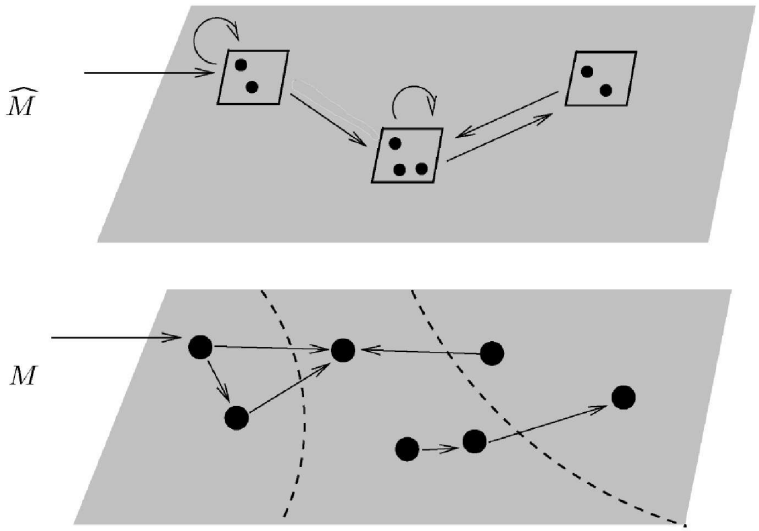
Existential abstraction

Existential abstraction



[Clarke, Grumberg, Jha, Lu, Veith 2003]

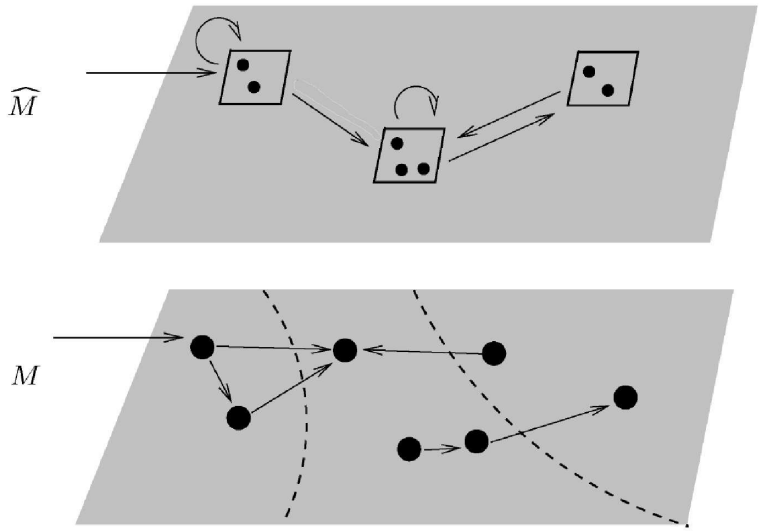
Existential abstraction



[Clarke, Grumberg, Jha, Lu, Veith 2003]

– Abstraction function $h : M \to \widehat{M}$ $\qquad (h : S \to \widehat{S})$

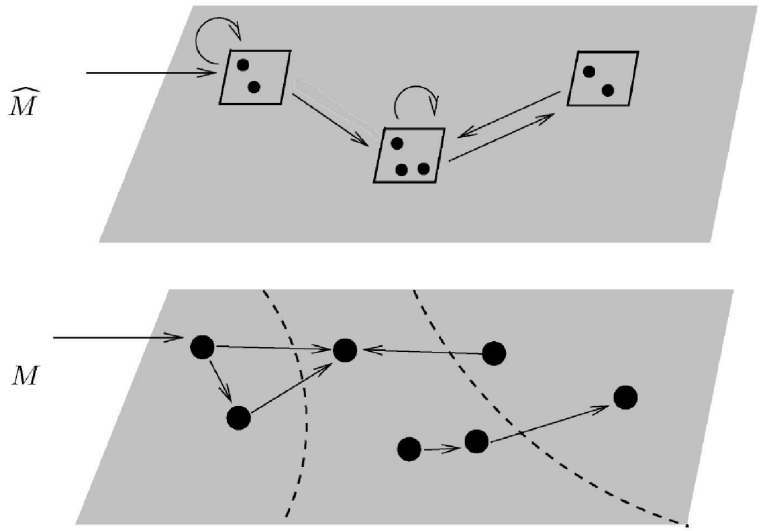
Existential abstraction



[Clarke, Grumberg, Jha, Lu, Veith 2003]

- Abstraction function $h : M \to \widehat{M}$ $\qquad (h : S \to \widehat{S})$
- Existential abstraction:
  - $\exists s \in S_0.\ h(s) = \widehat{s} \quad \Longrightarrow \quad \widehat{s} \in \widehat{S_0}$
  - $\exists s, s' \in S.\ R(s, s') \wedge h(s) = \widehat{s} \wedge h(s') = \widehat{s'} \quad \Longrightarrow \quad \widehat{R}(\widehat{s}, \widehat{s'})$

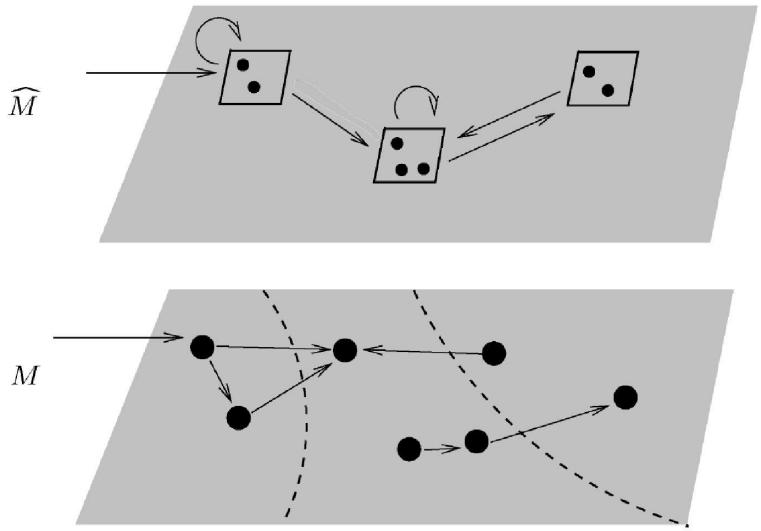
Existential abstraction



[Clarke, Grumberg, Jha, Lu, Veith 2003]

- Abstraction function $h : M \to \widehat{M}$ $\qquad (h : S \to \widehat{S})$
- Existential abstraction:
  - $\exists s \in S_0.\ h(s) = \widehat{s} \quad \Longrightarrow \quad \widehat{s} \in \widehat{S_0}$
  - $\exists s, s' \in S.\ R(s, s') \wedge h(s) = \widehat{s} \wedge h(s') = \widehat{s'} \quad \Longrightarrow \quad \widehat{R}(\widehat{s}, \widehat{s'})$
- We say that $\widehat{M}$ overapproximates M.

Minimal existential abstraction



[Clarke, Grumberg, Jha, Lu, Veith 2003]

- Abstraction function $h : M \to \widehat{M}$ $\qquad (h : s \to \widehat{S})$
- Minimal existential abstraction:
  - $\exists s \in S_0.\, h(s) = \widehat{s} \iff \widehat{s} \in \widehat{S_0}$
  - $\exists s, s' \in S.\, R(s, s') \wedge h(s) = \widehat{s} \wedge h(s') = \widehat{s'} \iff \widehat{R}(\widehat{s}, \widehat{s'})$
- We say that $\widehat{M}$ overapproximates M.

Model

Hardware: $M = (S, S_0, L, R)$ $V = \{x_1, \ldots, x_n\}$ $S = D^n$

R, S_0, L represented by formulas

abstraction applies to representation of the model

Software: abstraction applies to program source

running example: predicate abstraction

Predicate abstraction – basic idea

Example: Predicate $x > 0$

$$h_{x>0} : D \to \{\texttt{true}, \texttt{false}\}$$

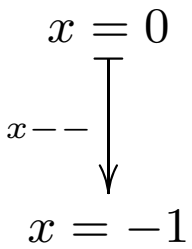
$$h_{x>0}(d) = \begin{cases} \texttt{true} & d > 0 \\ \texttt{false} & d \leq 0 \end{cases}$$

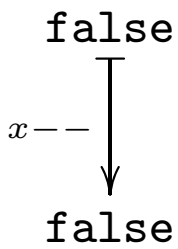
Predicate abstraction – basic idea

Example: Predicate $x > 0$

$$h_{x>0} : D \to \{\texttt{true}, \texttt{false}\}$$

$$h_{x>0}(d) = \begin{cases} \texttt{true} & d > 0 \\ \texttt{false} & d \leq 0 \end{cases}$$



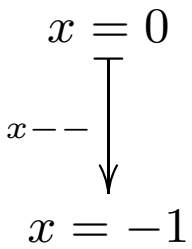


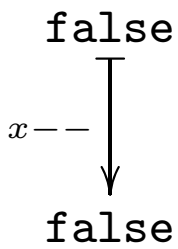
Predicate abstraction – basic idea

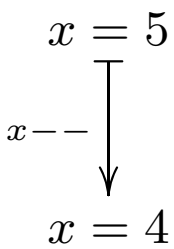
Example: Predicate $x > 0$

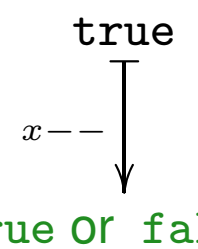
$$h_{x>0} : D \to \{\texttt{true}, \texttt{false}\}$$

$$h_{x>0}(d) = \begin{cases} \texttt{true} & d > 0 \\ \texttt{false} & d \leq 0 \end{cases}$$









nondeterminism!

Boolean programs

$h : D^n \rightarrow \{\texttt{true}, \texttt{false}\}^m$

```
    int x, y, z, w;

    void foo()
    {
[1]   do {
[2]     z = 0;
[3]     x = y;
[4]     if (w){
[5]       x++;
[6]       z = 1;
        }
[7]   } while(x!=y)
[8]   if(z){
[9]       assert(0);
      }
    }
```

```
    decl b1, b2;
   /* b1 stands for predicate (z=0) and
      b2 stands for predicate (x=y) */
    void foo()
    begin
[1]   do
[2]     b1 := 1;
[3]     b2 := 1;
[4]     if (*)
        begin
[5]       b2 := H(0,b2);
[6]       b1 := 0;
        end
[7]   while(b2)
[8]   if (!b1)
[9]     assert(0);
    end

    boolean H(e1,e2)
    begin
[10]  if (e1) then
[11]    return(1);
[12]  elsif (e2) then
[13]    return(0);
[14]  else
[15]    return(*);
      fi
   end
```

[T. Ball, S. K. Rajamani 2000]

Correctness via simulation

Simulation game $G_{M\ \widehat{M}}$

– players: Spoiler, Duplicator

Simulation game $G_{M\ \widehat{M}}$

– players: Spoiler, Duplicator

– configurations: $\langle s, \widehat{s} \rangle$

Simulation game $G_{M\ \widehat{M}}$

– players: Spoiler, Duplicator

– configurations: $\langle s, \widehat{s} \rangle$

– initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)

Simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins

Simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins
- a round
  - Spoiler chooses a transition $s \to r$ in M (assuming M total)
  - Duplicator answers with a transition $\widehat{s} \to \widehat{r}$ in $\widehat{M}$ (assuming $\widehat{M}$ total)
  - the play continues from the configuration $\langle r, \widehat{r} \rangle$

Simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins
- a round
  - Spoiler chooses a transition $s \to r$ in M (assuming M total)
  - Duplicator answers with a transition $\widehat{s} \to \widehat{r}$ in $\widehat{M}$ (assuming $\widehat{M}$ total)
  - the play continues from the configuration $\langle r, \widehat{r} \rangle$
- if the play is infinite, Duplicator wins

Simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins
- a round
  - Spoiler chooses a transition $s \to r$ in M (assuming M total)
  - Duplicator answers with a transition $\widehat{s} \to \widehat{r}$ in $\widehat{M}$ (assuming $\widehat{M}$ total)
  - the play continues from the configuration $\langle r, \widehat{r} \rangle$
- if the play is infinite, Duplicator wins

Question: How to adapt the definition for non-total models?

Simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins
- a round
  - Spoiler chooses a transition $s \to r$ in M (assuming M total)
  - Duplicator answers with a transition $\widehat{s} \to \widehat{r}$ in $\widehat{M}$ (assuming $\widehat{M}$ total)
  - the play continues from the configuration $\langle r, \widehat{r} \rangle$
- if the play is infinite, Duplicator wins

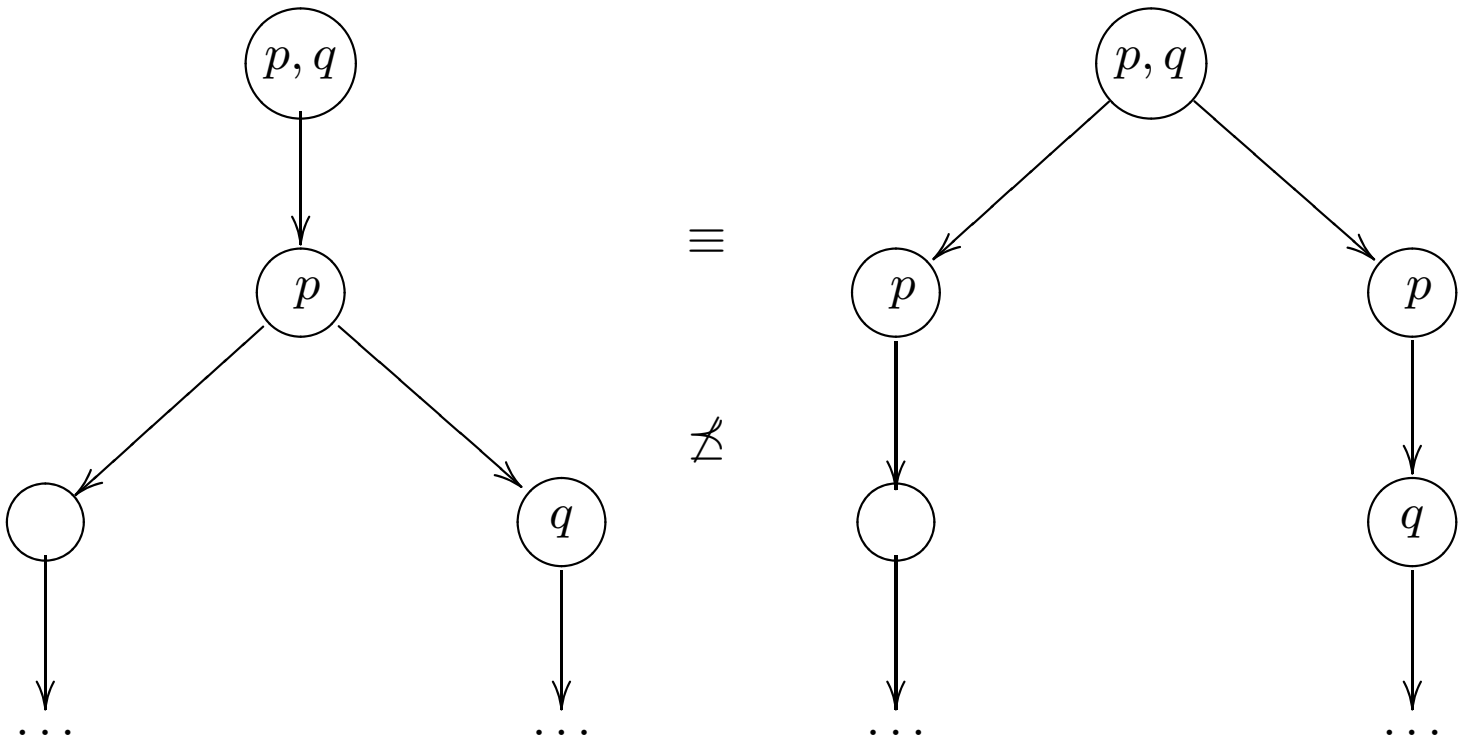
Question: How to adapt the definition for non-total models?

Question: How to adapt the definition for models with more initial states?

Simulation pre-order

Def.: $M \preceq \widehat{M} \iff$ Duplicator has a winning strategy in the game $G_{M\ \widehat{M}}$

Example:



Correctness via simulation

Simulation preorder is conservative:

Thm.: $M \preceq \widehat{M} \implies (\forall \phi \in \mathrm{ACTL}^*.\ \widehat{M} \models \phi \implies M \models \phi)$

Correctness via simulation

Simulation preorder is conservative:

Thm.: $M \preceq \widehat{M} \implies (\forall \phi \in \mathrm{ACTL}^*.\ \widehat{M} \models \phi \implies M \models \phi)$

Thm.: $M \preceq \widehat{M} \implies (\forall \phi \in \mathrm{ECTL}^*.\ M \models \phi \implies \widehat{M} \models \phi)$

Correctness via simulation

Simulation preorder is conservative:

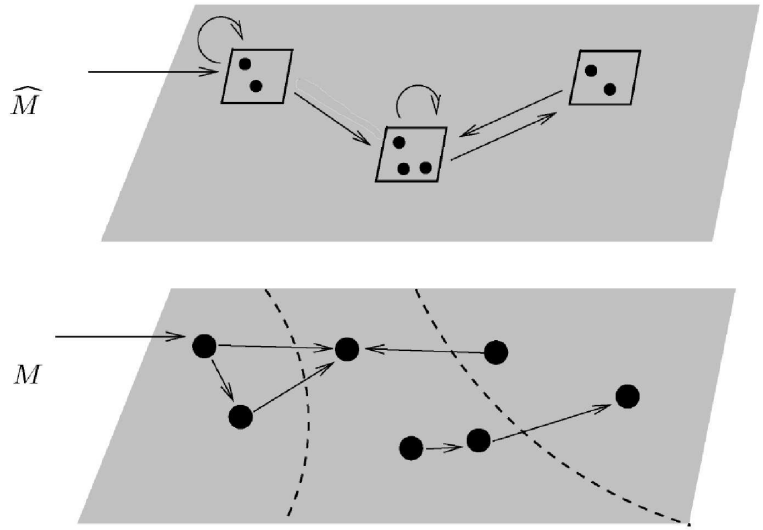
Thm.: $M \preceq \widehat{M} \implies (\forall \phi \in \mathrm{ACTL}^*.\ \widehat{M} \models \phi \implies M \models \phi)$

Thm.: $M \preceq \widehat{M} \implies (\forall \phi \in \mathrm{ECTL}^*.\ M \models \phi \implies \widehat{M} \models \phi)$

Def.: Simulation equivalence: $\simeq \ = \ \preceq \cap \succeq$

Thm.: $M \simeq \widehat{M} \implies (\forall \phi \in \mathrm{ACTL}^*.\ \widehat{M} \models \phi \iff M \models \phi)$

Correctness of existential abstraction



[Clarke, Grumberg, Jha, Lu, Veith 2003]

Abstraction function $h : M \to \widehat{M}$

Thm.: $M \preceq \widehat{M}$, if h preserves the atomic properties

$$h(L(s)) = \widehat{L}(h(s))$$

Fair simulation game $G_{M\ \widehat{M}}$

– players: Spoiler, Duplicator

– configurations: $\langle s, \widehat{s} \rangle$

– initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)

– if $L(s) \neq L(\widehat{s})$, Spoiler wins

Fair simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins
- a round
  - Spoiler chooses a fair path $s \to s_1 \to s_2 \to \ldots$ (assuming M total)
  - Duplicator answers with a fair path $\widehat{s} \to \widehat{s_1} \to \widehat{s_2} \to \ldots$
  - Spoiler chooses i
  - the play continues from the configuration $\langle s_i, \widehat{s_i} \rangle$

Fair simulation game $G_{M\ \widehat{M}}$

- players: Spoiler, Duplicator
- configurations: $\langle s, \widehat{s} \rangle$
- initial configuration: $\langle s_0, \widehat{s_0} \rangle$ (assuming one initial state in M and $\widehat{M}$)
- if $L(s) \neq L(\widehat{s})$, Spoiler wins
- a round
  - Spoiler chooses a fair path $s \to s_1 \to s_2 \to \ldots$ (assuming M total)
  - Duplicator answers with a fair path $\widehat{s} \to \widehat{s_1} \to \widehat{s_2} \to \ldots$
  - Spoiler chooses i
  - the play continues from the configuration $\langle s_i, \widehat{s_i} \rangle$
- if the play is infinite, Duplicator wins

Summary

$M \preceq \widehat{M}$	$\forall \phi \in \mathrm{ACTL}^*.\ \widehat{M} \models \phi \implies M \models \phi$
$M \preceq_F \widehat{M}$	$\forall \phi \in \mathrm{ACTL}^*.\ \widehat{M} \models_F \phi \implies M \models_F \phi$
$M \simeq \widehat{M}$	$\forall \phi \in \mathrm{ACTL}^*.\ M \models \phi \iff \widehat{M} \models \phi$
$M \simeq_F \widehat{M}$	$\forall \phi \in \mathrm{ACTL}^*.\ M \models_F \phi \iff \widehat{M} \models_F \phi$
$M \sim \widehat{M}$	$\forall \phi \in \mathrm{CTL}^*.\ M \models \phi \iff \widehat{M} \models \phi$
$M \sim_F \widehat{M}$	$\forall \phi \in \mathrm{CTL}^*.\ M \models_F \phi \iff \widehat{M} \models_F \phi$
$L_\omega(M) \subseteq L_\omega(\widehat{M})$	$\forall \phi \in \mathrm{LTL}.\ \widehat{M} \models \phi \implies M \models \phi$
$L_\omega(M) = L_\omega(\widehat{M})$	$\forall \phi \in \mathrm{LTL}.\ M \models \phi \iff \widehat{M} \models \phi$

CEGAR

CEGAR

Counter-example guided abstraction refinement

(1) Compute initial abstraction

CEGAR

Counter-example guided abstraction refinement

(1) Compute initial abstraction

(2) Repeat:

- verify the abstract model
  - if the result positive – answer yes and stop
  - if the result negative – abstract counterexample

CEGAR

Counter-example guided abstraction refinement

(1) Compute initial abstraction

(2) Repeat:

- verify the abstract model
  - if the result positive – answer yes and stop
  - if the result negative – abstract counterexample
- check feasibility of the counterexample
  - if the counterexample feasible – answer no and stop

CEGAR

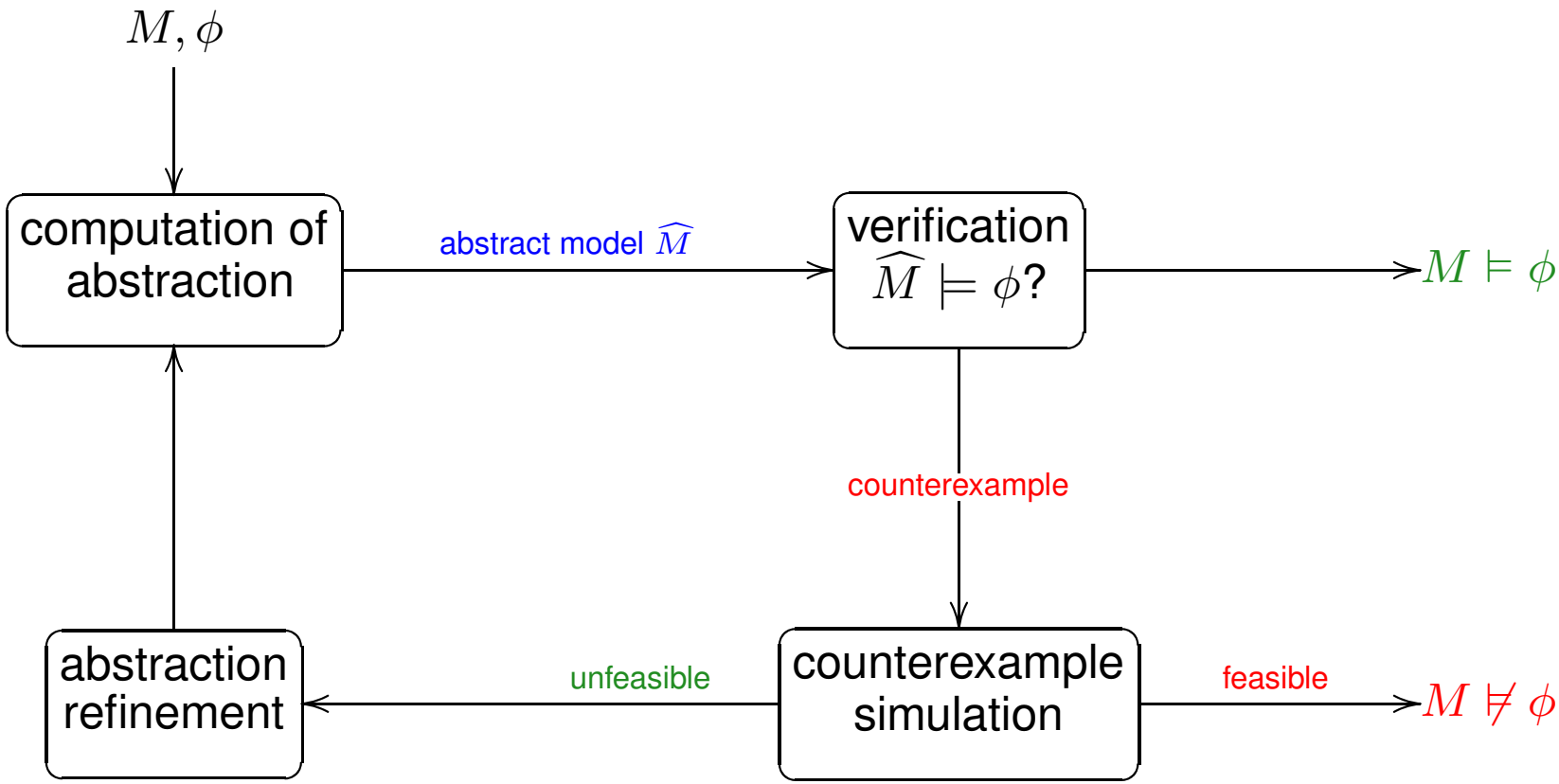
Counter-example guided abstraction refinement

(1) Compute initial abstraction

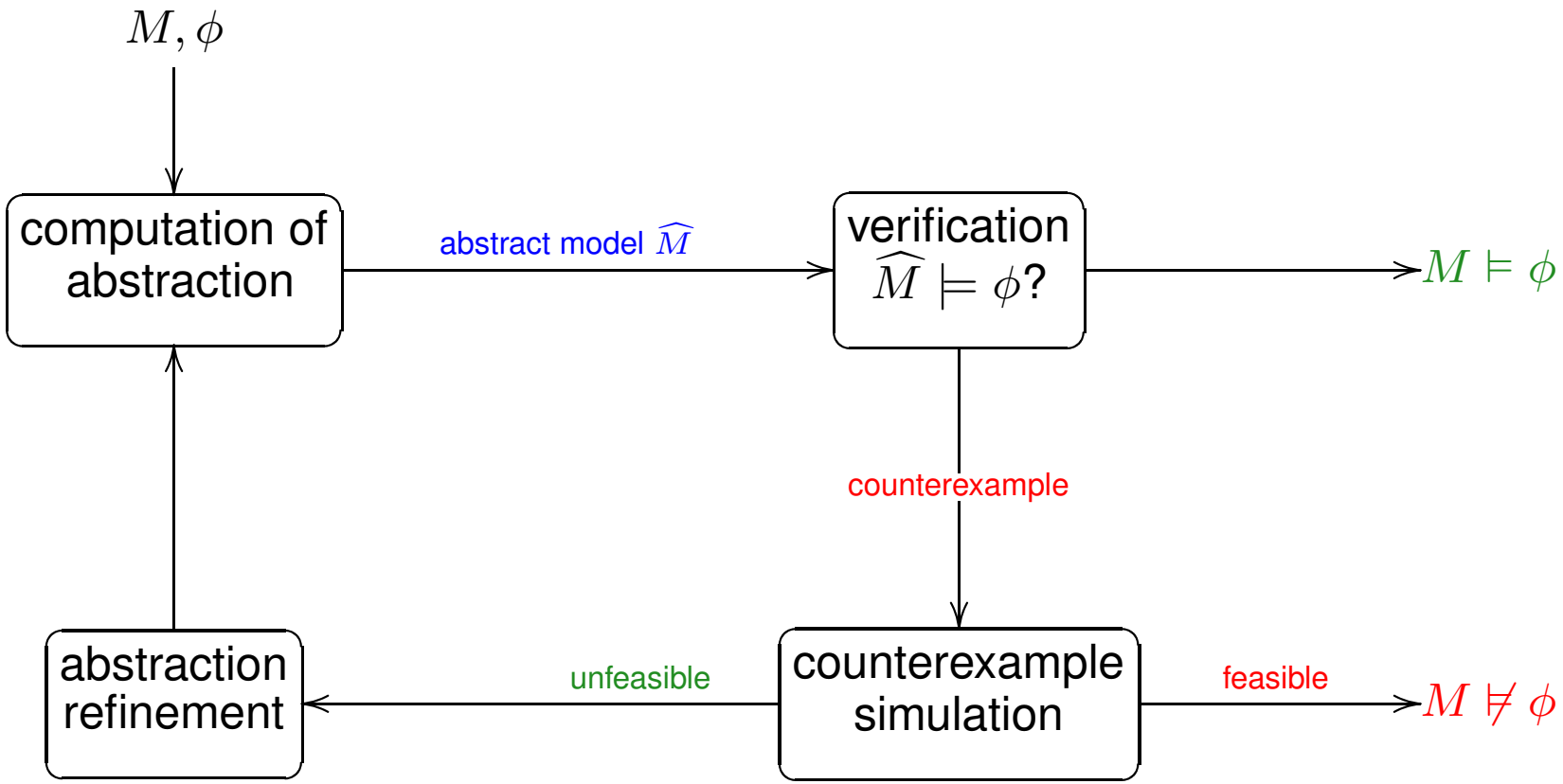
(2) Repeat:

- verify the abstract model
  - if the result positive – answer yes and stop
  - if the result negative – abstract counterexample
- check feasibility of the counterexample
  - if the counterexample feasible – answer no and stop
- refine the abstraction to eliminate the unfeasible counterexample

CEGAR loop

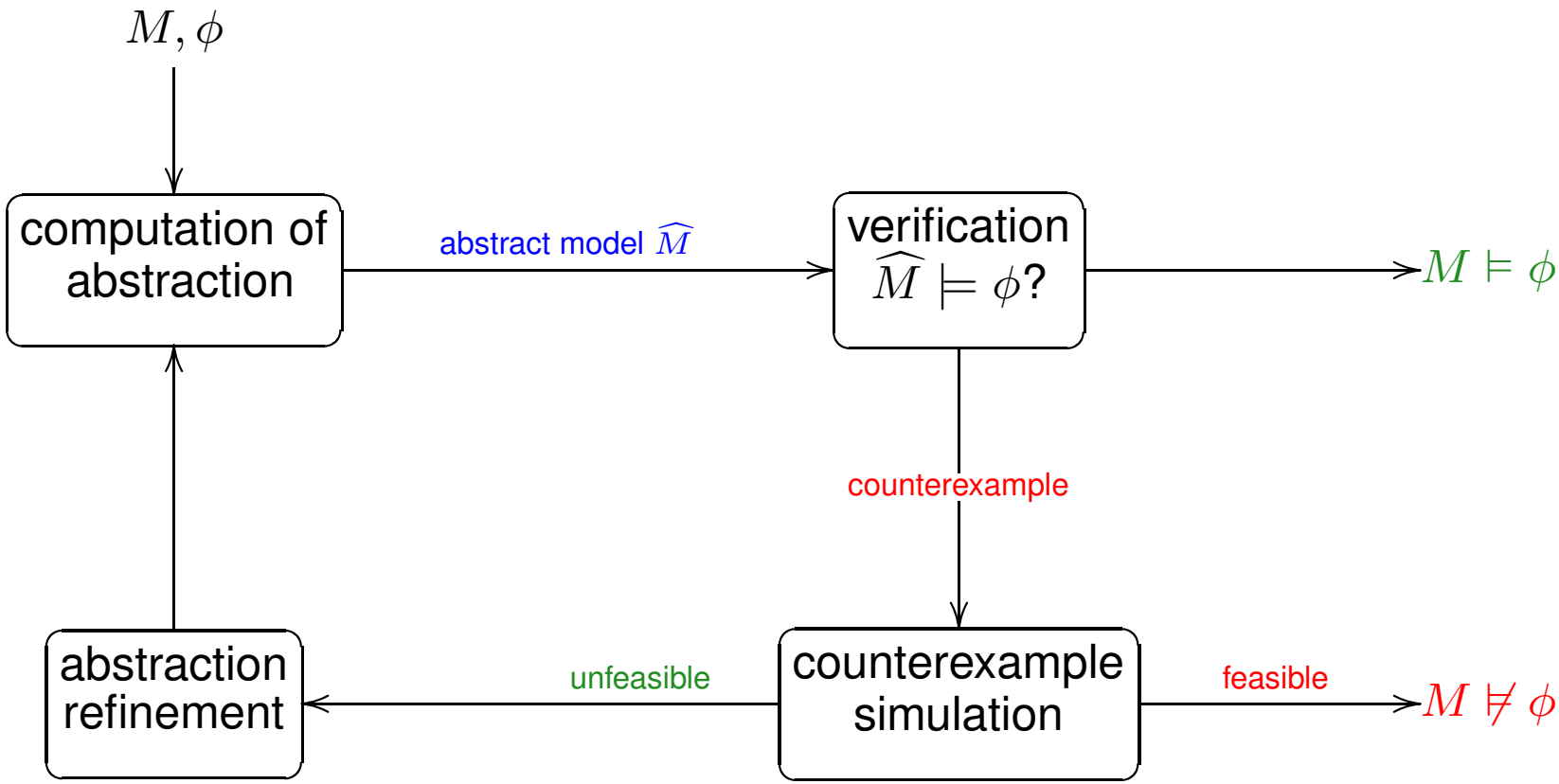

M, ϕ
computation of abstraction
abstract model $\widehat{M}$
verification $\widehat{M} \models \phi$?
$M \models \phi$
counterexample
abstraction refinement
unfeasible
counterexample simulation
feasible
$M \not\models \phi$

CEGAR loop



– correctness: whenever CEGAR stops the answer is correct

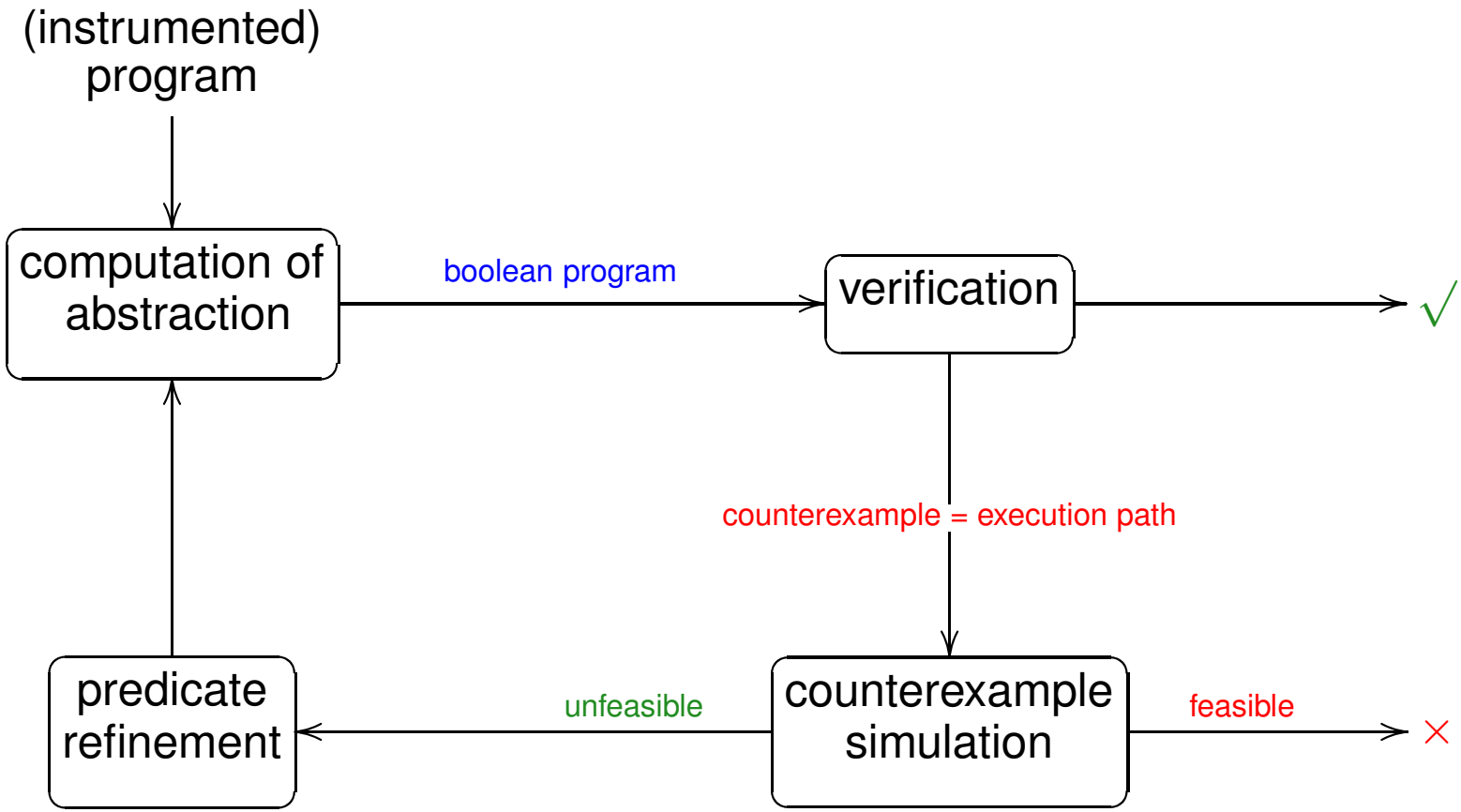
CEGAR loop



– correctness: whenever CEGAR stops the answer is correct

– termination: complete for finite systems, if minimal abstraction is used

predicate abstraction in CEGAR



CEGAR: initial abstraction

```
   numUnits: int;
   level: int;
   void getUnit() {
[1]   canEnter: bool := F;
[2]   if (numUnits = 0) {
[3]     if (level > 10){
[4]       NewUnit();
[5]       numUnits := 1;
[6]       canEnter := T;
        }
      } else
[7]     canEnter := T;

[8]   if (canEnter)
[9]     if (numUnits = 0)
[10]      assert(F);
        else
[11]      gotUnit();
   }
```

P

```
   void getUnit() {
[1]   ...;
[2]   if (?) {
[3]     if (?){
[4]       ...;
[5]       ...;
[6]       ...;
        }
      } else
[7]     ...;

[8]   if (?)
[9]     if (?)
[10]      ...;
        else
[11]      ...;
   }
```

B_1

[T. Ball, S. K. Rajamani 2000]

CEGAR example

```
   numUnits: int;
   level: int;
   void getUnit() {
[1]  canEnter: bool := F;
[2]  if (numUnits = 0) {
[3]    if (level > 10){
[4]      NewUnit();
[5]      numUnits := 1;
[6]      canEnter := T;
       }
     } else
[7]    canEnter := T;

[8]  if (canEnter)
[9]    if (numUnits = 0)
[10]     assert(F);
       else
[11]     gotUnit();
   }
```

P

```
   void getUnit() {
[1]  ...;
[2]  if (?) {
[3]    if (?){
[4]      ...;
[5]      ...;
[6]      ...;
       }
     } else
[7]    ...;

[8]  if (?)
[9]    if (?)
[10]     ...;
       else
[11]     ...;
   }
```

B_1

```
   nU0: bool;

   void getUnit() {
[1]  ...;
[2]  if (nU0) {
[3]    if (?){
[4]      ...;
[5]      nU0:=F;
[6]      ...;
       }
     } else
[7]    ...;

[8]  if (?){
[9]    if (nU0)
[10]     ...;
       else
[11]     ...;
   }
```

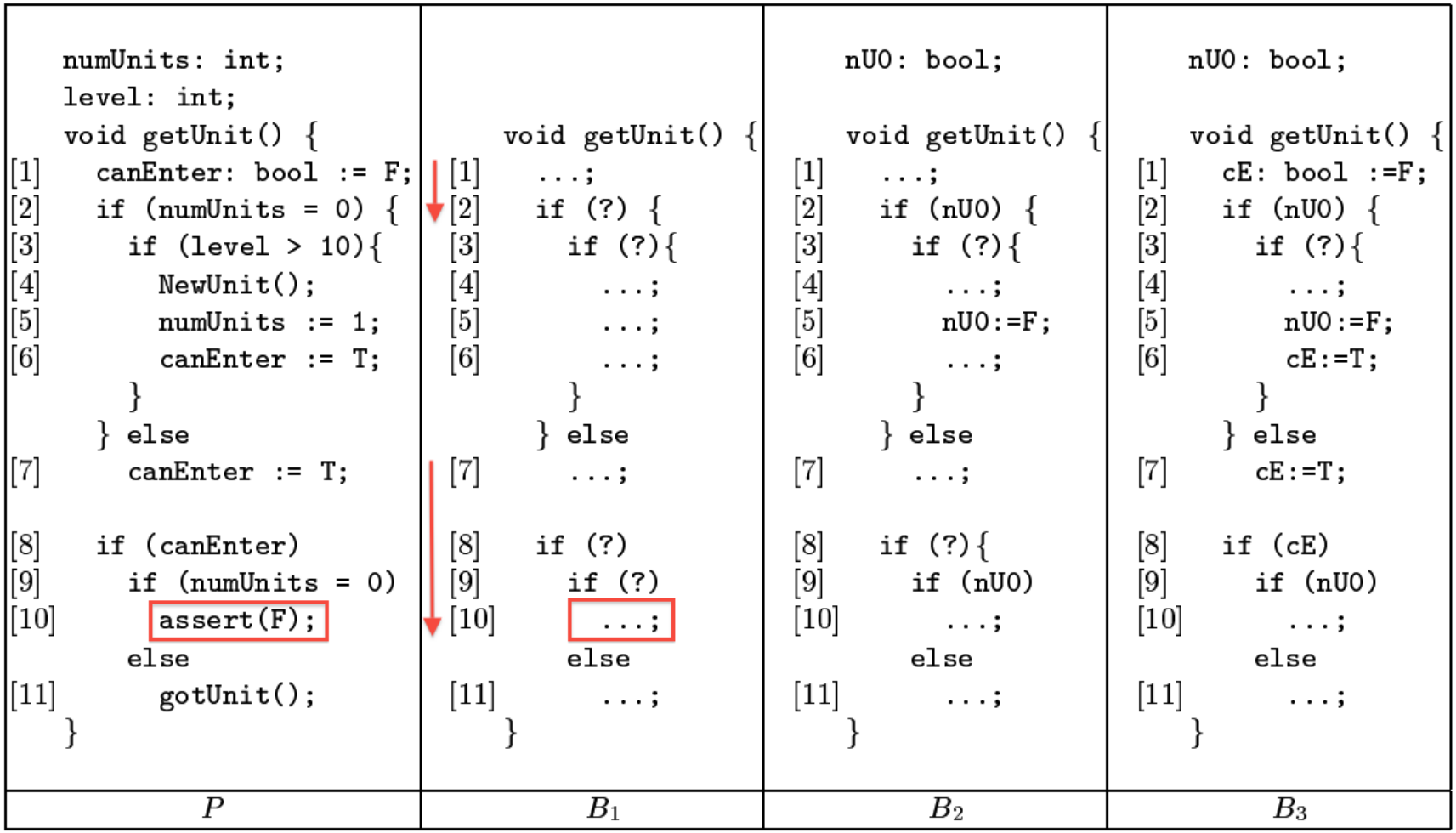
B_2

```
   nU0: bool;

   void getUnit() {
[1]  cE: bool :=F;
[2]  if (nU0) {
[3]    if (?){
[4]      ...;
[5]      nU0:=F;
[6]      cE:=T;
       }
     } else
[7]    cE:=T;

[8]  if (cE)
[9]    if (nU0)
[10]     ...;
       else
[11]     ...;
   }
```

B_3

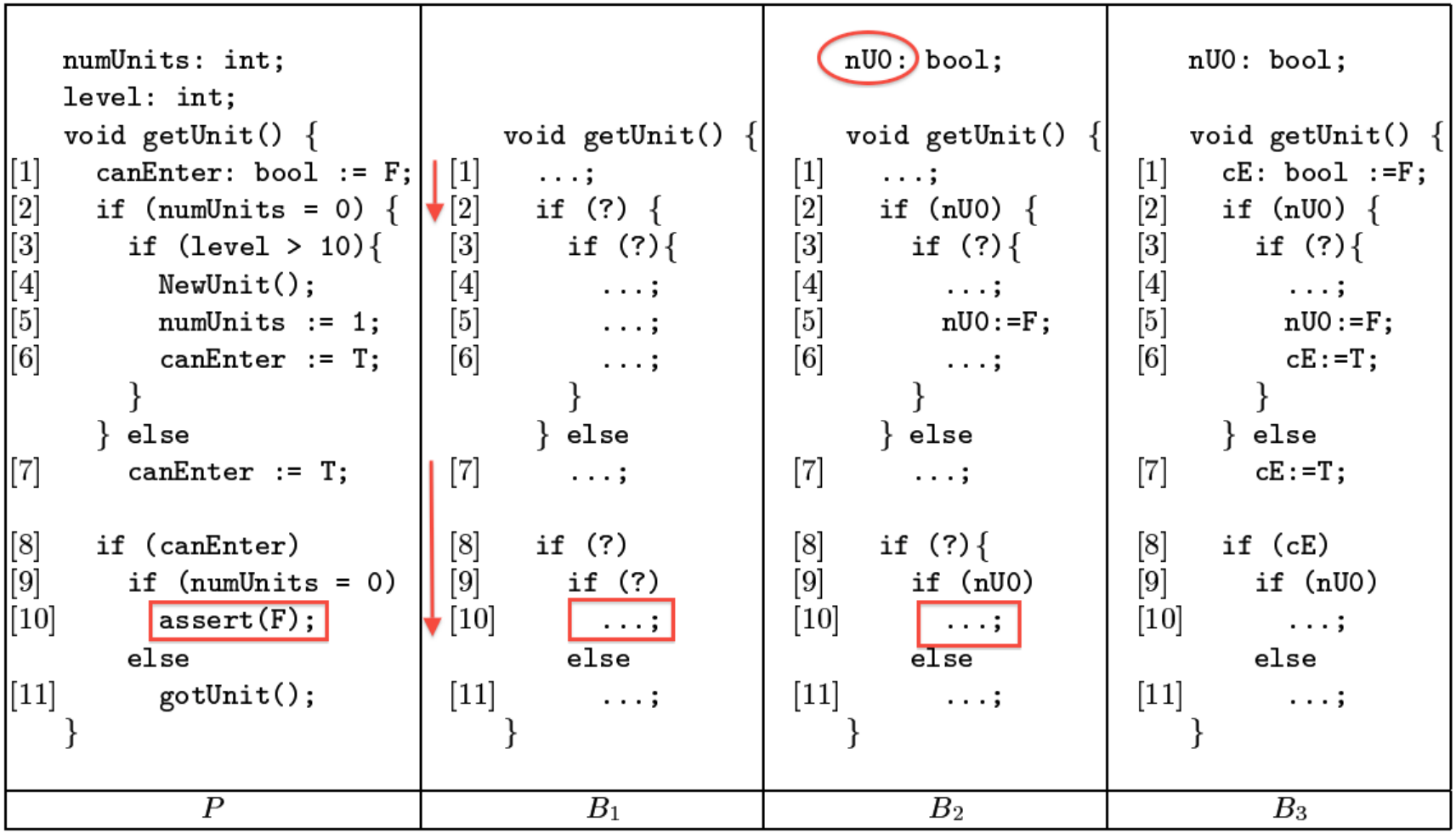
[T. Ball, S. K. Rajamani 2000]

CEGAR example



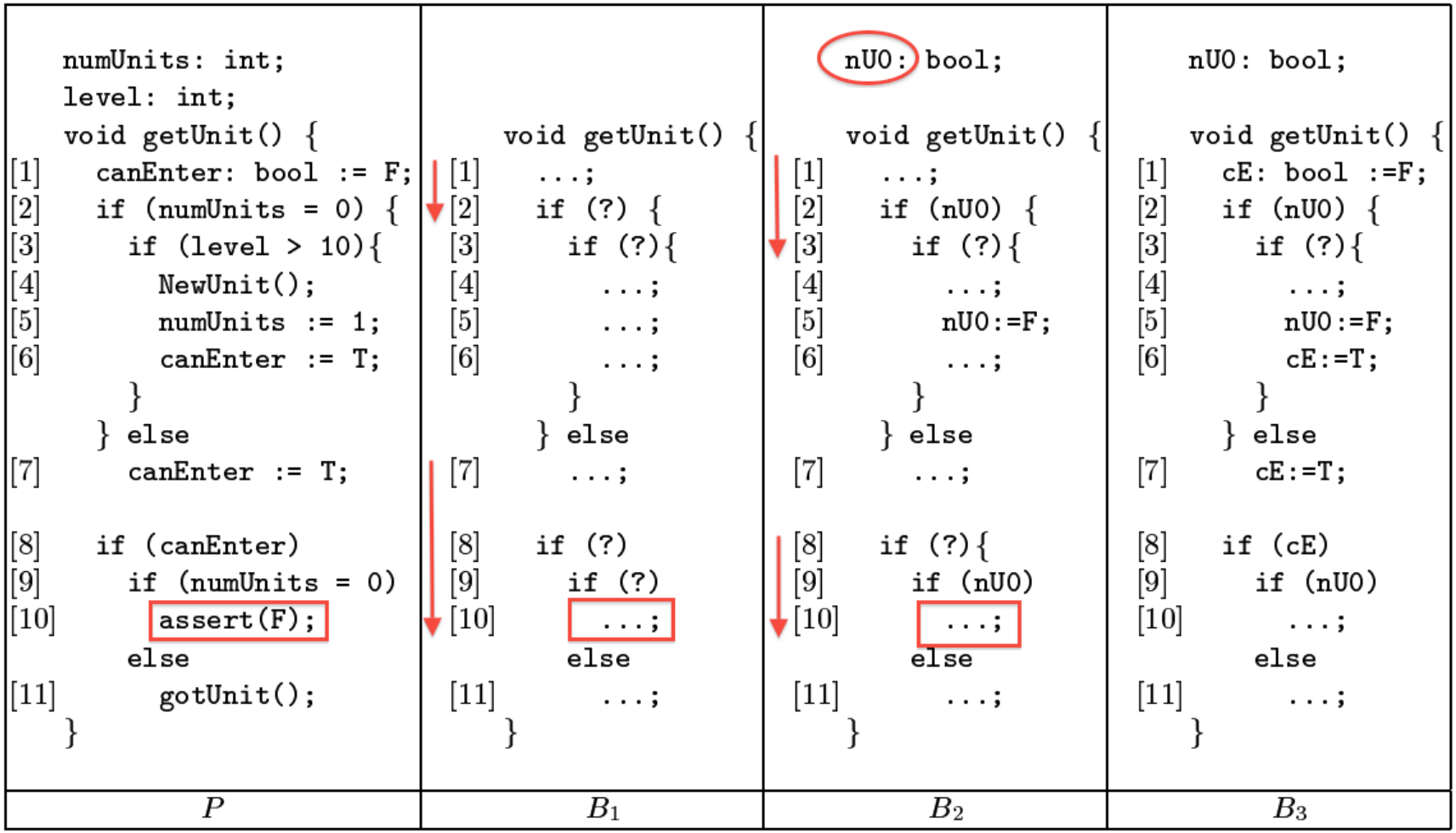
[T. Ball, S. K. Rajamani 2000]

CEGAR example



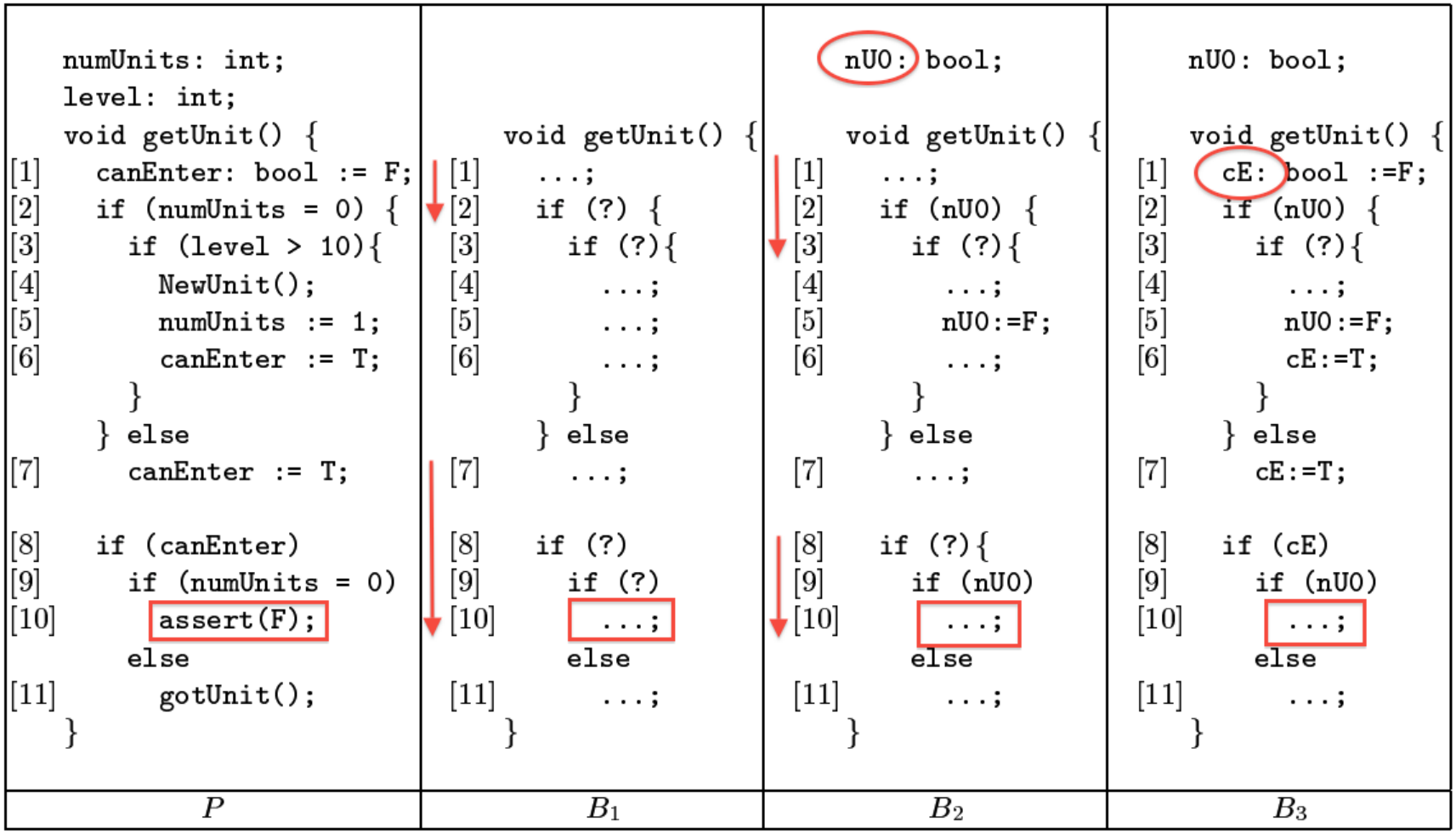
[T. Ball, S. K. Rajamani 2000]

CEGAR example



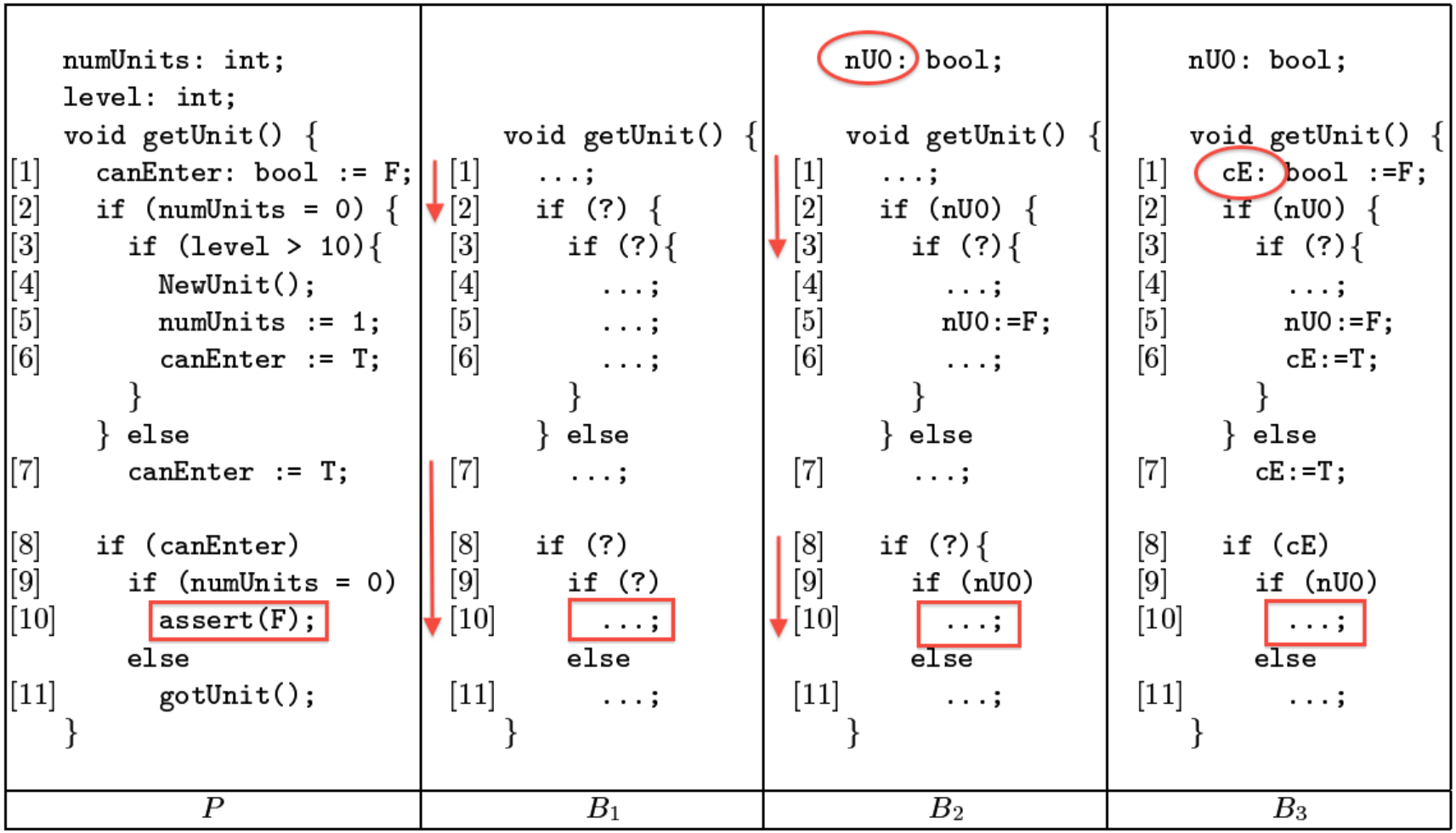
[T. Ball, S. K. Rajamani 2000]

CEGAR example



[T. Ball, S. K. Rajamani 2000]

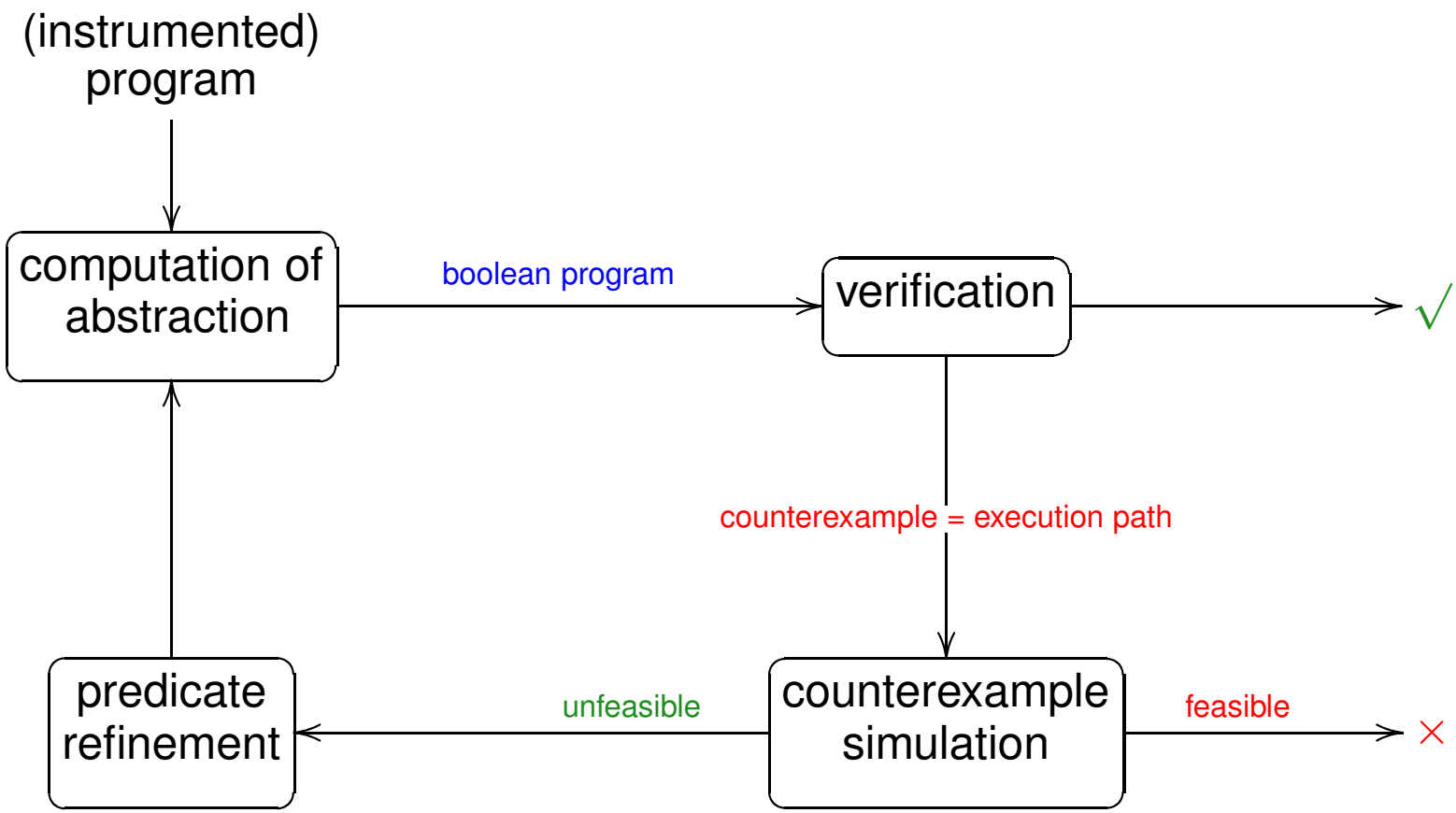
CEGAR example



[T. Ball, S. K. Rajamani 2000]

no reference to variable `level` was needed!

CEGAR: computation of abstraction



CEGAR: computation of abstraction

$$h : D^m \to \{\texttt{true}, \texttt{false}\}^n$$

CEGAR: computation of abstraction

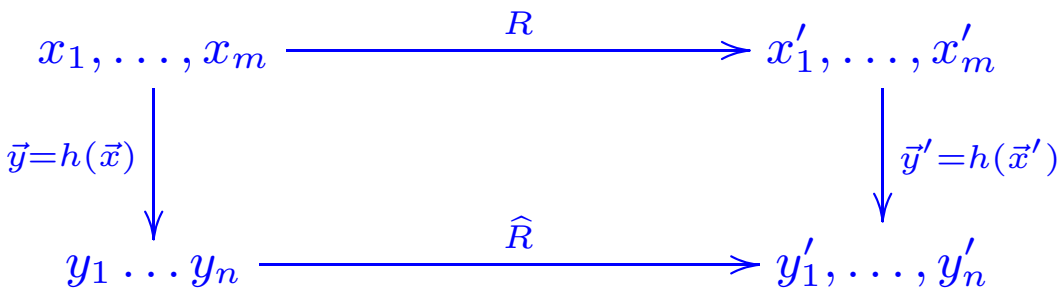
$$h : D^m \to \{\texttt{true}, \texttt{false}\}^n$$

– $\widehat{S} = \{\texttt{true}, \texttt{false}\}^n$

CEGAR: computation of abstraction

$$h : D^m \to \{\texttt{true}, \texttt{false}\}^n$$

- $\widehat{S} = \{\texttt{true}, \texttt{false}\}^n$
- $\widehat{R}(\vec{y}, \vec{y}') \equiv \exists \vec{x}, \vec{x}' .\, R(\vec{x}, \vec{x}') \wedge \vec{y}{=}h(\vec{x}) \wedge \vec{y}'{=}h(\vec{x}')$

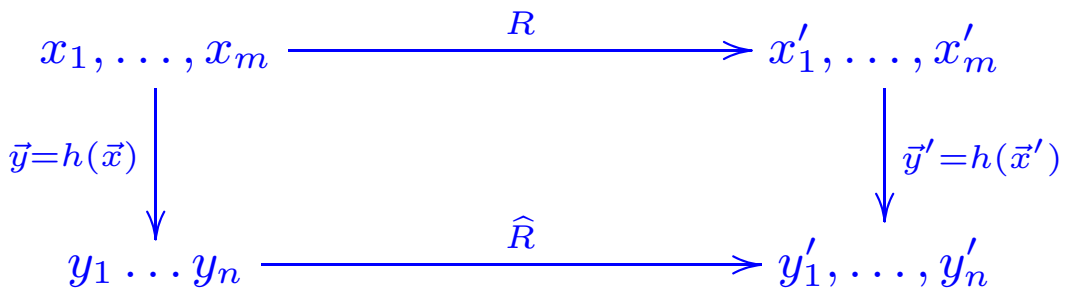


CEGAR: computation of abstraction

$$h : D^m \to \{\texttt{true}, \texttt{false}\}^n$$

– $\widehat{S} = \{\texttt{true}, \texttt{false}\}^n$

– $\widehat{R}(\vec{y}, \vec{y}') \equiv \exists \vec{x}, \vec{x}'.\, R(\vec{x}, \vec{x}') \wedge \vec{y}{=}h(\vec{x}) \wedge \vec{y}'{=}h(\vec{x}')$



– $\widehat{S_0}(\vec{y}) \equiv \exists \vec{x}.\, S_0(\vec{x}) \wedge \vec{y}{=}h(\vec{x})$

CEGAR: computation of abstraction

– simple block: `d := e; e++` $\mapsto$ $d' = e \wedge e' = e + 1$

CEGAR: computation of abstraction

– simple block: `d := e; e++` $\mapsto$ $d' = e \wedge e' = e + 1$

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

CEGAR: computation of abstraction

- simple block: `d := e; e++` $\mapsto$ $d' = e \ \wedge \ e' = e + 1$
- predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$
- abstract transitions:

$$\widehat{R}(b_1, b_2, b_1', b_2') \quad \equiv \quad \begin{array}{l} \exists d_0, e_0, d_1, e_1. \\ d' = e \ \wedge \ e' = e + 1 \ \wedge \\ b_1 = (e \geq 0) \ \wedge \ b_2 = (e \leq 100) \ \wedge \\ b_1' = (e' \geq 0) \ \wedge \ b_2' = (e' \leq 100) \end{array}$$

b_1	b_2	b_1'	b_2'
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	0
1	1	1	1

CEGAR: computation of abstraction

- simple block: `d := e; e++` $\mapsto$ $d' = e \ \wedge \ e' = e + 1$
- predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$
- abstract transitions:

$$\widehat{R}(b_1, b_2, b_1', b_2') \quad \equiv \quad \begin{array}{l} \exists d_0, e_0, d_1, e_1. \\ d' = e \ \wedge \ e' = e + 1 \ \wedge \\ b_1 = (e \geq 0) \ \wedge \ b_2 = (e \leq 100) \ \wedge \\ b_1' = (e' \geq 0) \ \wedge \ b_2' = (e' \leq 100) \end{array}$$

b_1	b_2	b_1'	b_2'
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	0
1	1	1	1

- nondeterminism

CEGAR: computation of abstraction

– simple block: `d := e; e++` $\mapsto$ $d' = e \ \wedge \ e' = e + 1$

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

– abstract transitions:

$$\widehat{R}(b_1, b_2, b_1', b_2') \quad \equiv \quad \begin{array}{l} \exists d_0, e_0, d_1, e_1. \\ d' = e \ \wedge \ e' = e + 1 \wedge \\ b_1 = (e \geq 0) \ \wedge \ b_2 = (e \leq 100) \wedge \\ b_1' = (e' \geq 0) \ \wedge \ b_2' = (e' \leq 100) \end{array}$$

b_1	b_2	b_1'	b_2'
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	0
1	1	1	1

– nondeterminism

– calls to a prover or SMT-solver

CEGAR: computation of abstraction

– simple block: `d := e; e++` $\mapsto$ $d' = e \ \wedge \ e' = e + 1$

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

– abstract transitions:

$$\widehat{R}(b_1, b_2, b_1', b_2') \quad \equiv \quad \begin{array}{l} \exists d_0, e_0, d_1, e_1. \\ d' = e \ \wedge \ e' = e + 1 \ \wedge \\ b_1 = (e \geq 0) \ \wedge \ b_2 = (e \leq 100) \ \wedge \\ b_1' = (e' \geq 0) \ \wedge \ b_2' = (e' \leq 100) \end{array}$$

b_1	b_2	b_1'	b_2'
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	0
1	1	1	1

– nondeterminism

– calls to a prover or SMT-solver

– non-minimal abstraction (for minimal one, exponential number of calls)

CEGAR: computation of abstraction

– simple block: `d := e; e++` $\mapsto$ $d' = e \ \wedge \ e' = e + 1$

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

– abstract transitions:

$$\widehat{R}(b_1, b_2, b_1', b_2') \quad \equiv \quad \begin{array}{l} \exists d_0, e_0, d_1, e_1. \\ d' = e \ \wedge \ e' = e + 1 \wedge \\ b_1 = (e \geq 0) \ \wedge \ b_2 = (e \leq 100) \wedge \\ b_1' = (e' \geq 0) \ \wedge \ b_2' = (e' \leq 100) \end{array}$$

b_1	b_2	b_1'	b_2'
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	0
1	1	1	1

– nondeterminism

– calls to a prover or SMT-solver

– non-minimal abstraction (for minimal one, exponential number of calls)

– calls to SAT-solver (assuming boolean encoding of source program)

CEGAR: computation of abstraction

– iteratively call SAT-solver:
  – a satisfying valuation $\mapsto$ a blocking clause
  – bactracking to the highest decision level
  – the procedure stops when unsatisfiable

b_1	b_2	b_1'	b_2'
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	1	0
1	1	1	1

CEGAR: computation of abstraction

– control flow: `if(e > 50) {goto l} else {goto l'}`

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

CEGAR: computation of abstraction

– control flow: `if(e > 50) {goto l} else {goto l'}`

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

– abstract control flow: `if(`$b_1 \vee \neg b_2$`) {goto l}` +

`if(`$b_1 \vee b_2$`) {goto l'}`

CEGAR: computation of abstraction

– control flow: `if(e > 50) {goto l} else {goto l'}`

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

– abstract control flow: `if(`$b_1 \vee \neg b_2$`) {goto l}` +

`if(`$b_1 \vee b_2$`) {goto l'}`

– abstract guards not necessarily disjoint!

CEGAR: computation of abstraction

– control flow: `if(e > 50) {goto l} else {goto l'}`

– predicates: $b_1 = (e \geq 0), \quad b_2 = (e \leq 100)$

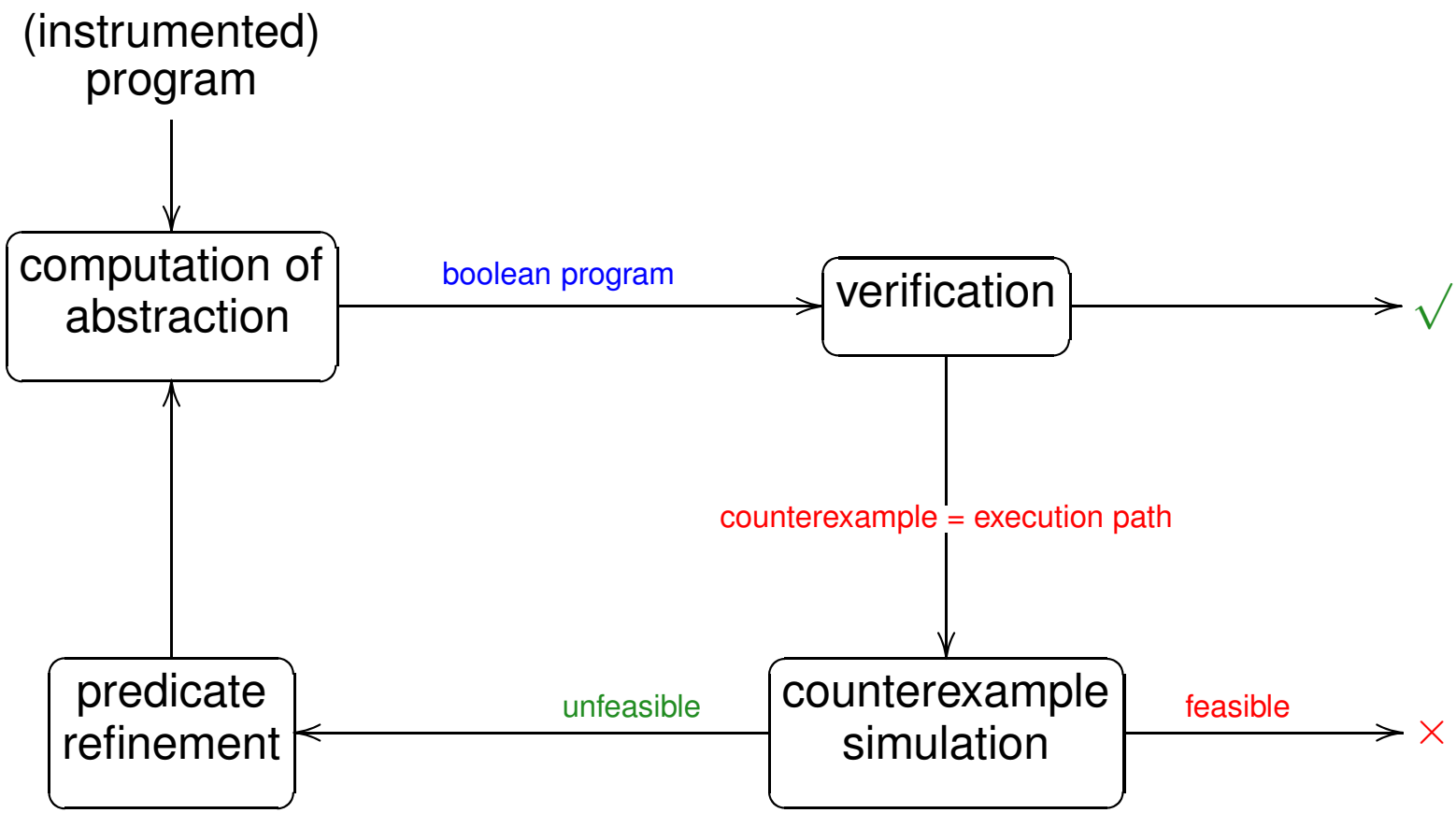
– abstract control flow: `if(`$b_1 \vee \neg b_2$`) {goto l}` `+`

`if(`$b_1 \vee b_2$`) {goto l'}`

– abstract guards not necessarily disjoint!

– typically `if` (`while`) guards are among predicates

CEGAR: verification



CEGAR: verification

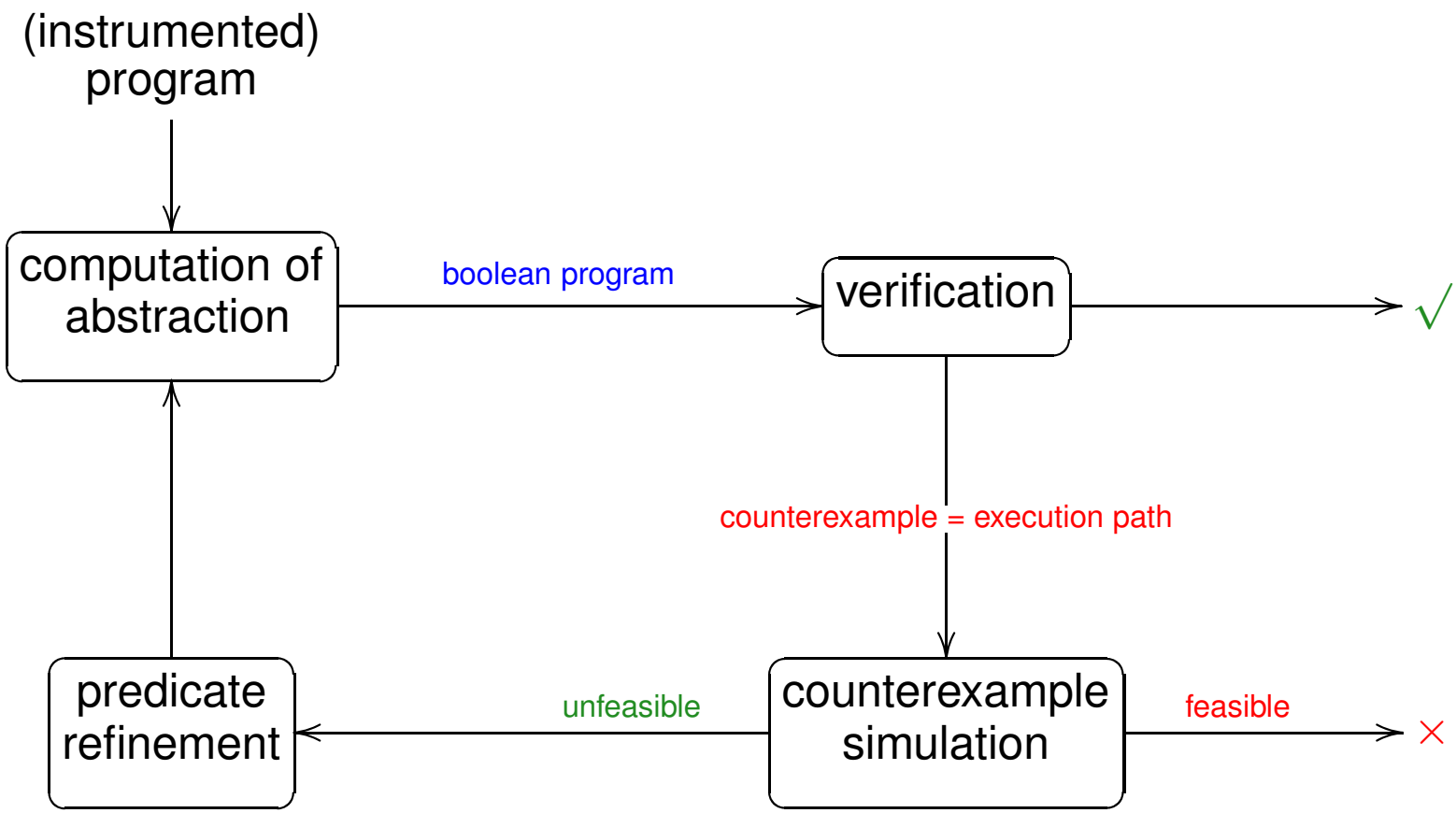
– SMC may be convenieniently used (if no recursion nor dynamic allocation)

$$
\begin{array}{llllllll}
 & \neg b_1 & \wedge & b_2 & \wedge & \neg b_1' & \wedge & b_2' \\
\vee & b_1 & \wedge & \neg b_2 & \wedge & \neg b_1' & \wedge & b_2' \\
\vee & b_1 & & & \wedge & b_1' & \wedge & \neg b_2' \\
\vee & & & b_2 & \wedge & b_1' & \wedge & b_2'
\end{array}
$$

```
TRANS PC=x -> (!b1 &  b2  & !next(b1) &  next(b2))
            | ( b1 & !b2  & !next(b1) &  next(b2))
            | ( b1        &  next(b1) & !next(b2))
            | (       b2  &  next(b1) &  next(b2))
```

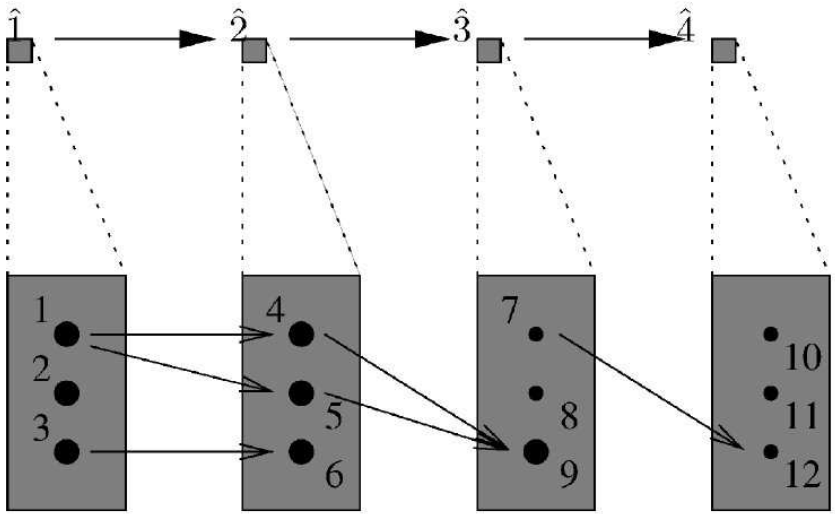
[Clarke, Kroening, Sharygina, Yorav 2004]

CEGAR: counterexample simulation



CEGAR: counterexample simulation

abstract counterexample $\widehat{1}\,\widehat{2}\,\widehat{3}\,\widehat{4}$



[Clarke, Grumberg, Jha, Lu, Veith 2003]

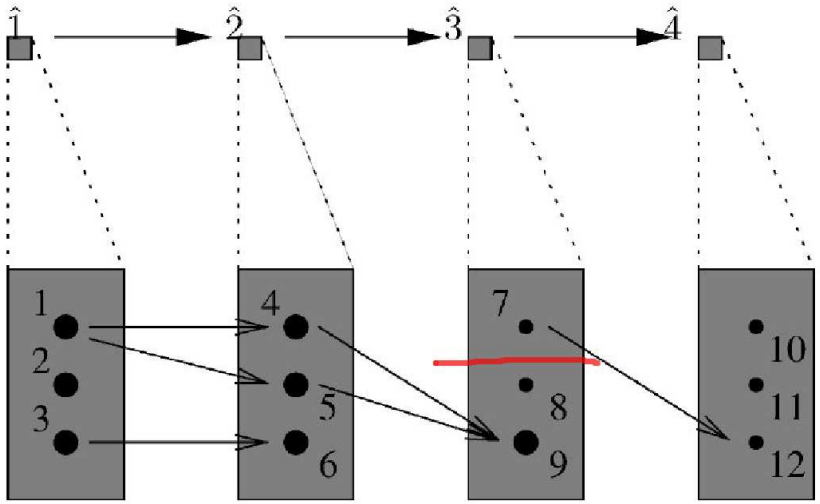
counterexample simulation (strongest post-condition)

$$S_1 := S_0 \cap h^{-1}(\,\widehat{1}\,) \qquad\qquad S_i := \vec{R}(S_{i-1}) \cap h^{-1}(\,\widehat{i}\,)$$

by iterative calls to a prover, SMT-solver, or SAT-solver

CEGAR: spurious counterexample

abstract counterexample $\widehat{1}\,\widehat{2}\,\widehat{3}\,\widehat{4}$

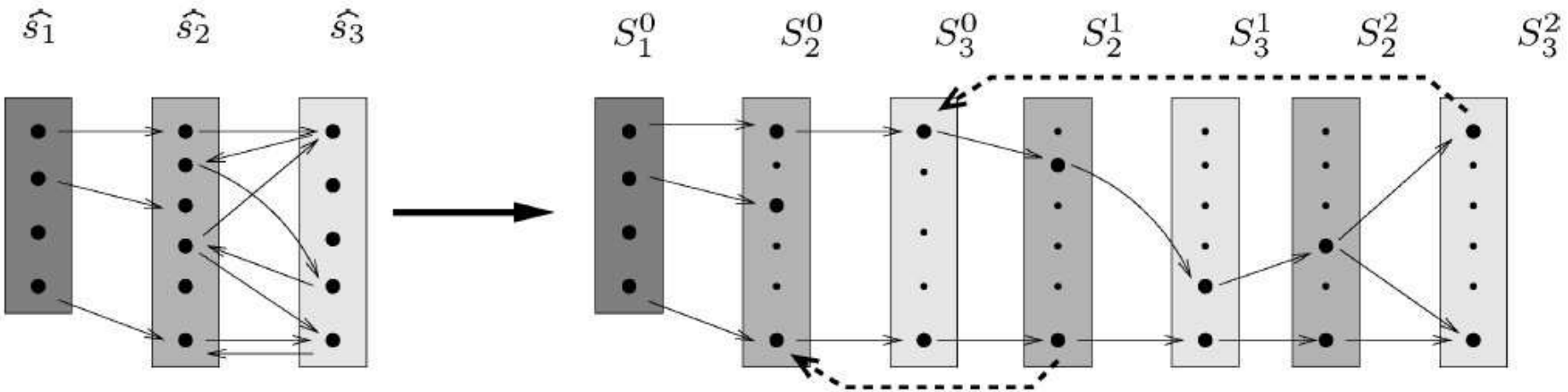


[Clarke, Grumberg, Jha, Lu, Veith 2003]

$$S_4 = \vec{R}(S_3) \cap h^{-1}(\,\widehat{4}\,) = \emptyset \qquad S_3 \;\cap\; \vec{R}^{-1}(h^{-1}(\,\widehat{4}\,)) = \emptyset$$

CEGAR: infinite counterexample

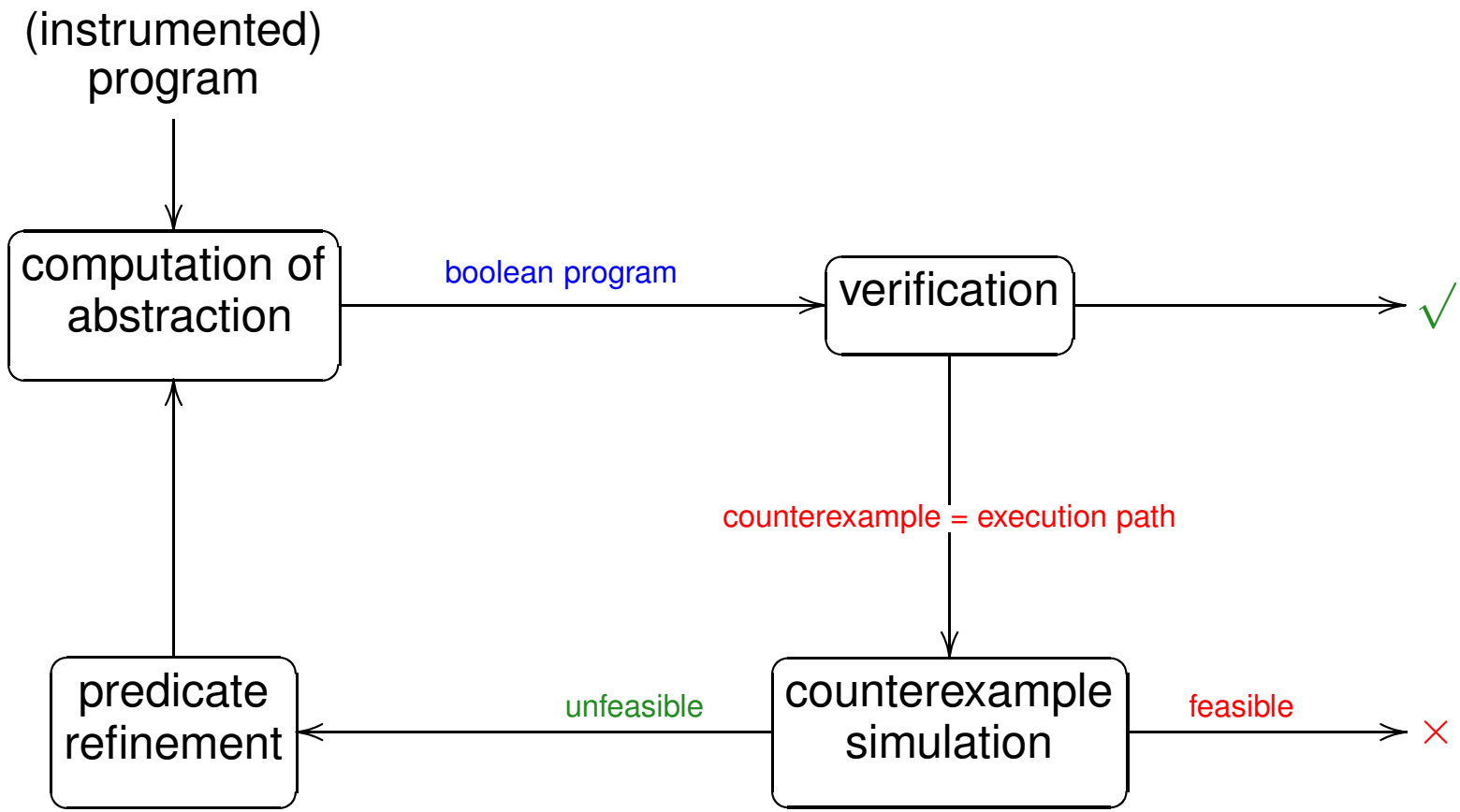
abstract counterexample $\widehat{s_1}(\widehat{s_2}\widehat{s_3})^\omega$



[Clarke, Grumberg, Jha, Lu, Veith 2003]

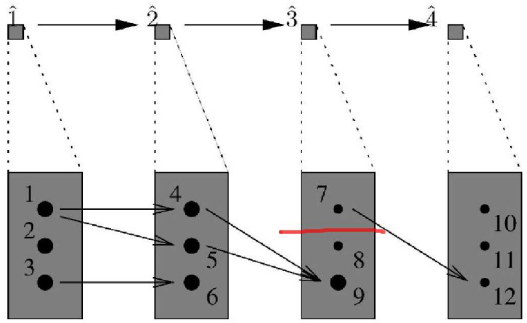
– unfold the loop $\min\{\text{size}(\widehat{s_2}), \text{size}(\widehat{s_3})\}$ times

CEGAR: predicate refinement



aim: elimination of the spurious counterexample

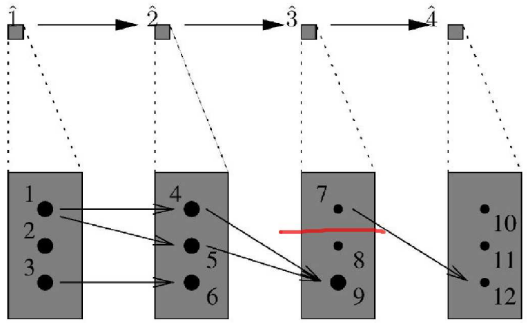
CEGAR: predicate refinement



– spurious counterexample:

$$R(\vec{x}_1, \vec{x}_2) \ \wedge \ R(\vec{x}_2, \vec{x}_3) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_n)$$ unsatisfiable

CEGAR: predicate refinement



– spurious counterexample:

$$R(\vec{x}_1, \vec{x}_2) \ \wedge \ R(\vec{x}_2, \vec{x}_3) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_n) \quad \text{unsatisfiable}$$

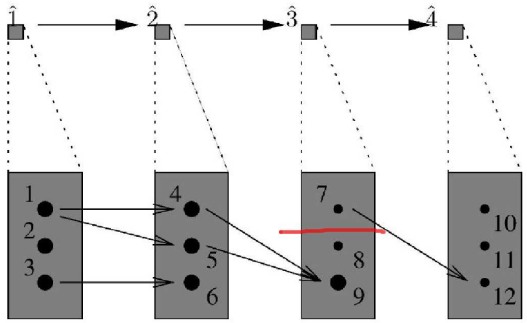
– Craig interpolation: exists a formula $\phi(\vec{x}_i)$ such that

  – $\phi(\vec{x}_i)$ only uses variables x_i

  – $R(\vec{x}_1, \vec{x}_2) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_i) \implies \phi$

  – $\phi \ \wedge \ R(\vec{x}_i, \vec{x}_{i+1}) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_n)$ unsatisfiable

CEGAR: predicate refinement



– spurious counterexample:

$$R(\vec{x}_1, \vec{x}_2) \ \wedge \ R(\vec{x}_2, \vec{x}_3) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_n) \quad \text{unsatisfiable}$$

– Craig interpolation: exists a formula $\phi(\vec{x}_i)$ such that

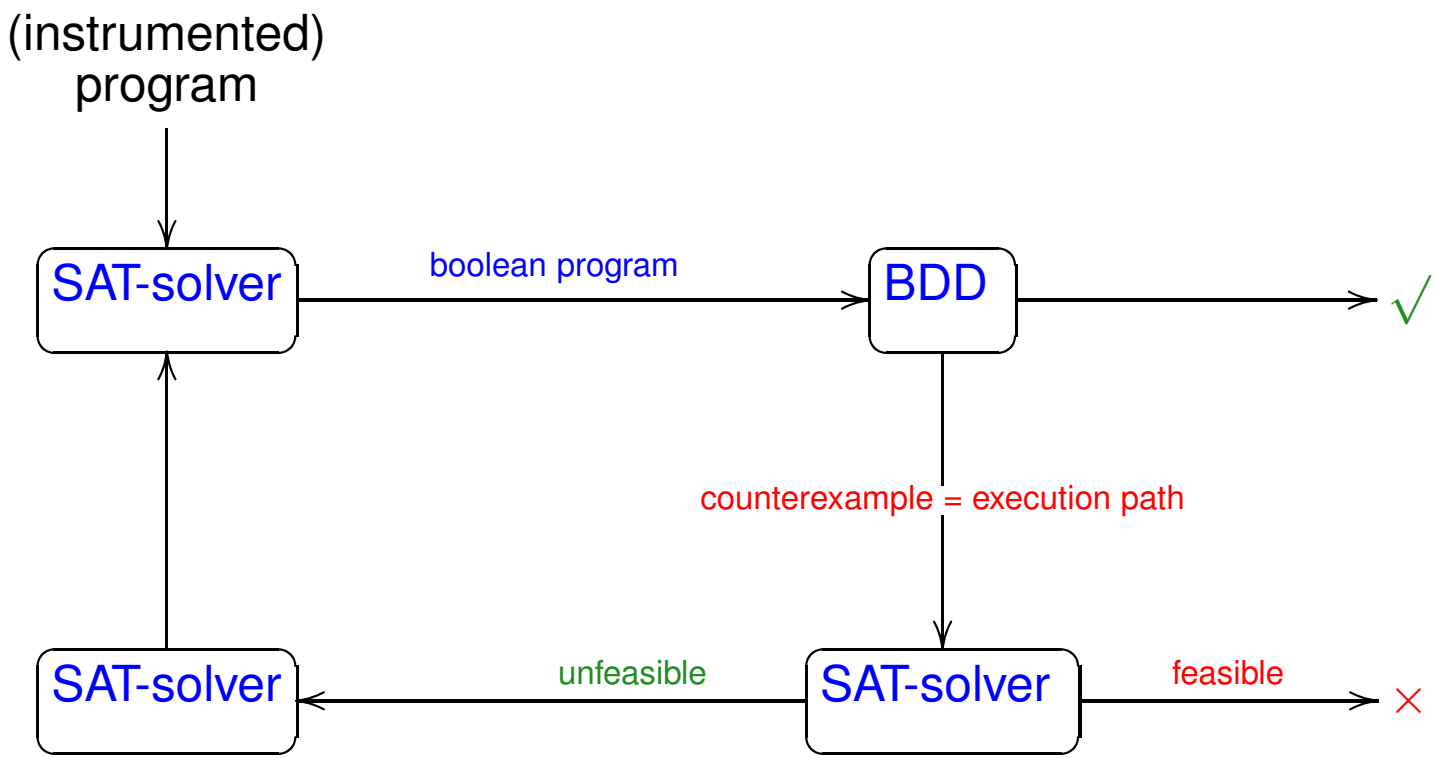
  – $\phi(\vec{x}_i)$ only uses variables x_i

  – $R(\vec{x}_1, \vec{x}_2) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_i) \implies \phi$

  – $\phi \ \wedge \ R(\vec{x}_i, \vec{x}_{i+1}) \ \wedge \ \ldots \ \wedge \ R(\vec{x}_{n-1}, \vec{x}_n)$ unsatisfiable

– For propositional logic, extract Craig interpolant from SAT-solver output

CEGAR: BDD + SAT



Software verification success story

– 85% of system crashes of Windows XP caused by bugs in third-party kernel-level device drivers (2003)

– one of reasons is the complexity of the Windows drivers API

– SLAM: automatically checks device drivers for certain correctness properties with respect to the Windows device drivers API

– now part of Windows Driver Development Kit, a toolset for drivers developers