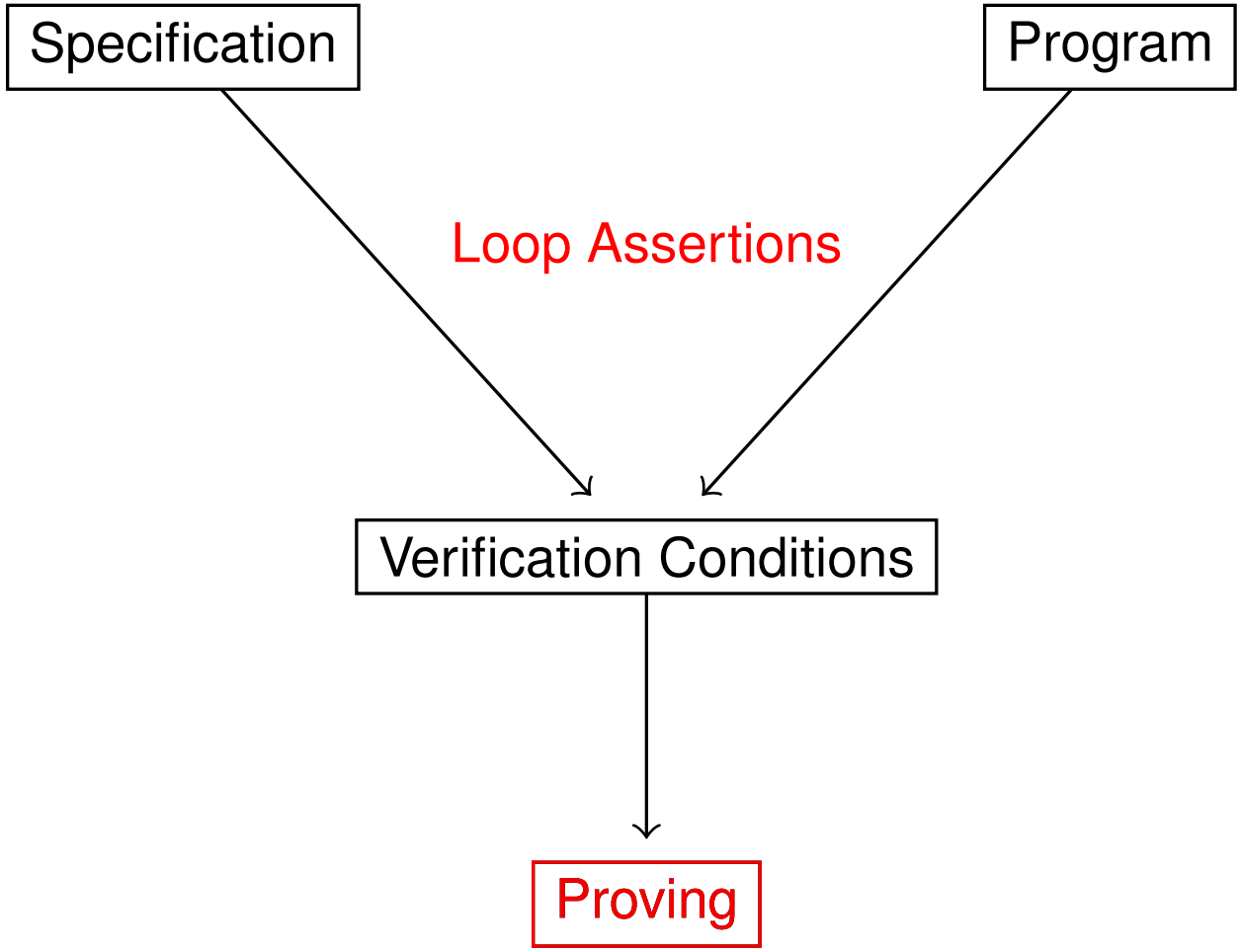
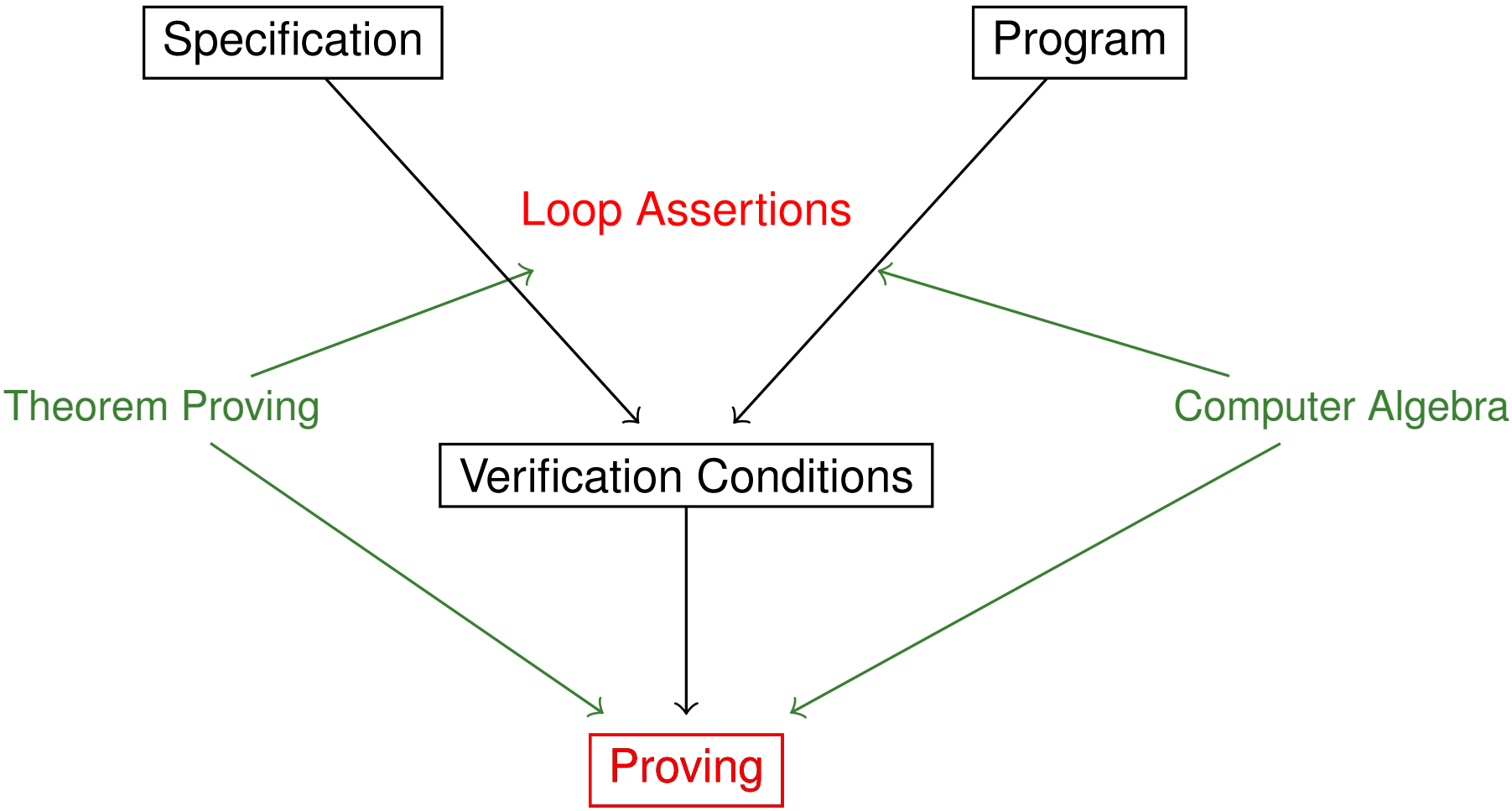
Symbolic Computation and Theorem Proving in Program Analysis

Laura Kovács

Chalmers


Specification
Program
Loop Assertions
Verification Conditions
Proving


Specification
Program
Loop Assertions
Theorem Proving
Computer Algebra
Verification Conditions
Proving

Assertion Synthesis — Example: Array Partition

Program

$a := 0;\ b := 0;\ c := 0;$

while $(a < N)$ **do**

  if $A[a] \geq 0$

    then $B[b] := A[a];\ b := b + 1$

    else $C[c] := A[a];\ c := c + 1;$

  $a := a + 1;$

end do

Loop Assertions

$a = b + c$

$a \geq 0 \ \wedge \ b \geq 0 \ \wedge \ c \geq 0$

$a \leq N \quad \vee \quad N \leq 0$

$(\forall p)(p \geq b \rightarrow B[p] = B_0[p])$

$(\forall p)(0 \leq p < b \rightarrow$

$\quad B[p] \geq 0 \wedge$

$\quad (\exists i)(0 \leq i < a \ \wedge \ A[a] = B[p]))$

Assertion Synthesis — Example: Array Partition

Program

$a := 0;\ b := 0;\ c := 0;$

while $(a < N)$ **do**

  if $A[a] \geq 0$

    then $B[b] := A[a];\ b := b + 1$

    else $C[c] := A[a];\ c := c + 1;$

  $a := a + 1;$

end do

Loop Assertions

Polynomial Equalities and Inequalities, Quantified FO properties

$$a = b + c$$

$$a \geq 0 \ \wedge \ b \geq 0 \ \wedge \ c \geq 0$$

$$a \leq N \quad \vee \quad N \leq 0$$

$$(\forall p)(p \geq b \rightarrow B[p] = B_0[p])$$

$$(\forall p)(0 \leq p < b \rightarrow$$
$$B[p] \geq 0 \wedge$$
$$(\exists i)(0 \leq i < a \ \wedge \ A[a] = B[p]))$$

Our Approach

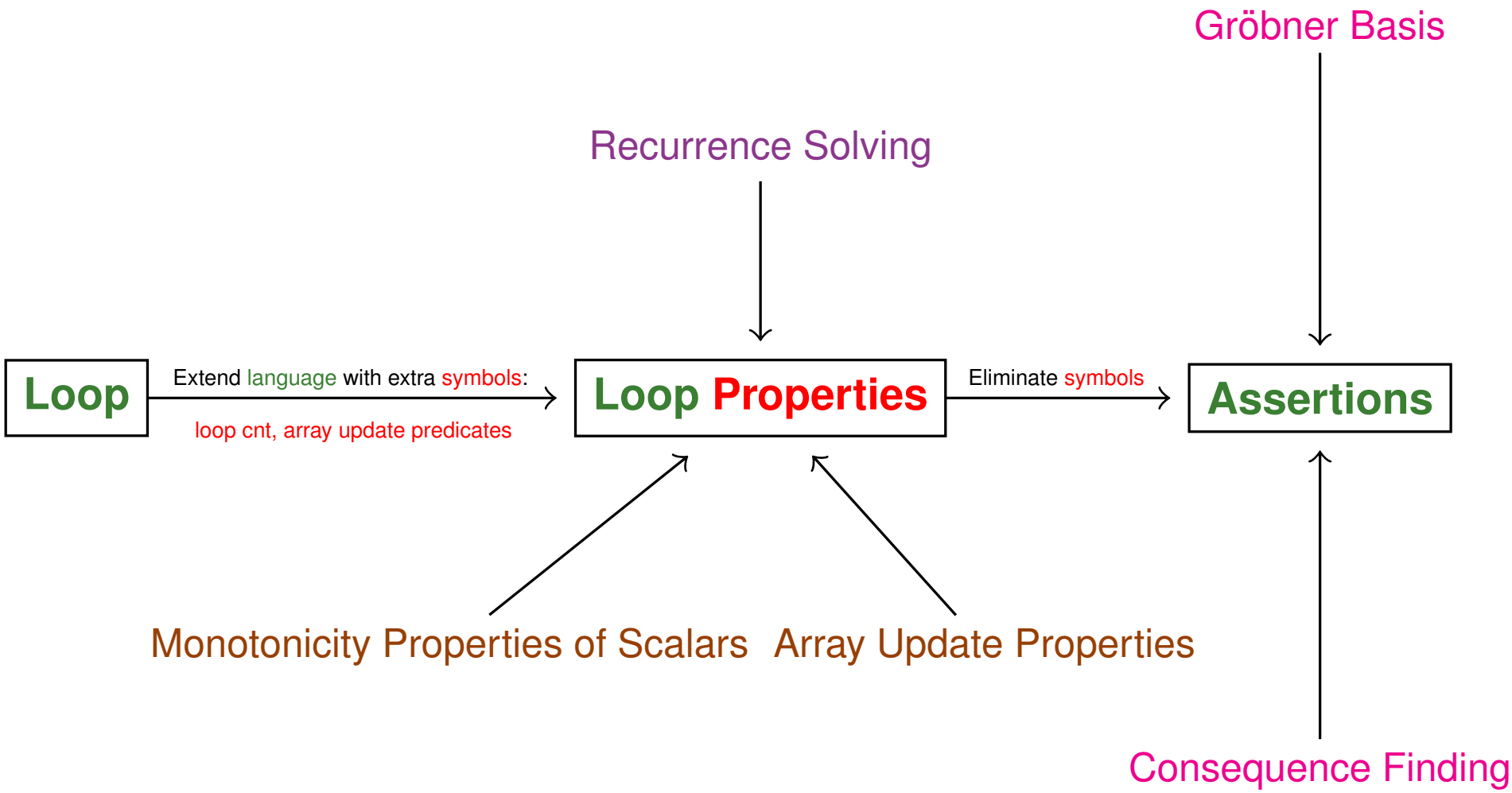
Loop

Assertions

Our Approach

Our Approach:
SYMBOL ELIMINATION

Our Approach:
SYMBOL ELIMINATION



Outline

Part 1: Weakest Precondition for Program Analysis and Verification

Part 2: Polynomial Invariant Generation (TACAS'08, LPAR'10)

Part 3: Quantified Invariant Generation (FASE'09, MICAI'11)

Part 4: Invariants, Interpolants and Symbol Elimination (CADE'09, POPL'12, APLAS'12)

Part 3: Quantified Invariant Generation

Symbol Elimination by First-Order Theorem Proving

Outline

Example: Array Partition partition.c

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 0$

B :

*	*	*	*	*	*	*

$b = 0$

C :

*	*	*	*	*	*	*

$c = 0$

Example: Array Partition partition.c

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Example: Array Partition partition.c

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

- Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$.

$$(\forall p)(0 \leq p < b \rightarrow B[p] \geq 0 \wedge (\exists i)(0 \leq i < a \wedge A[i] = B[p]))$$

Example: Array Partition partition.c

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

- Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$.

$$(\forall p)(0 \leq p < b \rightarrow B[p] \geq 0 \wedge (\exists i)(0 \leq i < a \ \wedge \ A[i] = B[p]))$$

Example: Array Partition partition.c

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

A :

1	3	-1	-5	8	0	-2

$a = 7$

B :

1	3	8	0	*	*	*

$b = 4$

C :

-1	-5	-2	*	*	*	*

$c = 3$

Invariants with $\forall$ $\exists$

- Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$.
- Each of $C[0], \ldots, C[c-1]$ is negative and equal to one of $A[0], \ldots, A[a-1]$.

Invariants with $\forall$

- For every $p \geq b$, the value of $B[p]$ is equal to its initial value.
- For every $p \geq c$, the value of $C[p]$ is equal to its initial value.

Example: Array Partition - Some Experiments

$a := 0; b := 0; c := 0;$
while $(a \leq k)$ **do**
  if $A[a] \geq 0$
    then $B[b] := A[a]; b := b + 1;$
    else $C[c] := A[a]; c := c + 1;$
  $a := a + 1;$
end do

1. B doesn't change at positions after final value of b ($1s$):

$$\forall p(p \geq b \rightarrow B[p] = B_0[p])$$

2. Each $B[0], \ldots, B[b-1]$ is a positive value in $\{A[0], \ldots, A[a-1]\}$ ($1s$):

$$\forall p(b > p \wedge p \geq 0 \rightarrow B[p] \geq 0 \wedge \exists k(a > k \wedge k \geq 0 \wedge A[k] = B[p])$$

Outline

Overview of the Method

- Given loop $\mathcal{L}$;
- Extend $\mathcal{L}$ to $\mathcal{L}'$;
- Extract a set P of loop properties in $\mathcal{L}'$;
- Generate loop property p in $\mathcal{L}$ s.t. $P \rightarrow p$.

Overview of the Method

- Given loop $\mathcal{L}$;
- Extend $\mathcal{L}$ to $\mathcal{L}'$;
- Extract a set P of loop properties in $\mathcal{L}'$;
- Generate loop property p in $\mathcal{L}$ s.t. $P \rightarrow p$.

Overview of the Method

- Given loop $\mathcal{L}$;
- Extend $\mathcal{L}$ to $\mathcal{L}'$;
- Extract a set P of loop properties in $\mathcal{L}'$;
- Generate loop property p in $\mathcal{L}$ s.t. $P \rightarrow p$.

  $\leftarrow$ Symbol elimination!

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

- $upd_V(i,p)$: at iteration i, V is updated at position p;
- $upd_V(i,p,x)$: at iteration i, V is updated at position p by value x.

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

- Increasing/decreasing (strictly): $(\forall i \in iter)(v^{(i+1)} \geq v^{(i)})$
- Dense: $(\forall i \in iter)(v^{(i+1)} = v^{(i)} \vee v^{(i+1)} = v^{(i)} + 1)$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

- Strictly increasing and dense: $(\forall i \in iter)(v^{(i)} = v^{(0)} + i)$
- Increasing but not strictly increasing: $(\forall j,k \in iter)(k \geq j \rightarrow v^{(k)} \geq v^{(j)})$
- Increasing, dense and not strictly increasing: $(\forall j,k \in iter)(k \geq j \rightarrow v^{(j)} + k \geq v^{(k)} + j)$

$v^{(k)} \geq v^{(j)} \xrightarrow{\text{dense}} v^{(j)} + k - j \geq v^{(k)}$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

- Update predicates of (d.i.) variables:
  $(\forall p)(v^{(0)} \leq p < v^{(n)} \rightarrow (\exists i \in iter)(\bigvee_{u \in U}(i :: G_u) \wedge v^{(i)} = p)$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
     - Stability
       no array update at $p \rightarrow$ (final) value of $V[p]$ is unchanged
     - Last update
       $V[p]$ updated at iteration i and no further $\rightarrow$ final value $V[p]$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

3. Eliminate symbols $\rightarrow$ Invariants

Invariant Generation - The Method

```
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

1. Extend the language $\mathcal{L}$ to $\mathcal{L}'$:
   - variables as functions of n: $v^{(i)}$ with $0 \leq i < n$
   - predicates as loop properties: $iter, upd_V(i,p), upd_V(i,p,x)$

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$
$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$
$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

2. Collect loop properties:
   - Polynomial scalar properties
   - Monotonicity properties of scalars
   - Update predicates of arrays
   - Translation of guarded assignments

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$
$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$
$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$
$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$
$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$
$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$
$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$
$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

3. Eliminate symbols $\rightarrow$ Invariants

HOW?

Invariant Generation by Symbol Elimination

$$(\forall i)(i \in iter \Leftrightarrow 0 \leq i \wedge i < n)$$

$$upd_B(i,p) \Leftrightarrow i \in iter \wedge p = b^{(i)} \wedge A[a^{(i)}] \geq 0$$

$$upd_B(i,p,x) \Leftrightarrow upd_B(i,p) \wedge x = A[a^{(i)}]$$

$$a = b + c,\ a \geq 0,\ b \geq 0,\ c \geq 0$$

$$(\forall i \in iter)(a^{(i+1)} > a^{(i)})$$

$$(\forall i \in iter)(b^{(i+1)} = b^{(i)} \vee b^{(i+1)} = b^{(i)} + 1)$$

$$(\forall i \in iter)(a^{(i)} = a^{(0)} + i)$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(k)} \geq b^{(j)})$$

$$(\forall j,k \in iter)(k \geq j \rightarrow b^{(j)} + k \geq b^{(k)} + j)$$

$$(\forall p)(b^{(0)} \leq p < b^{(n)} \rightarrow (\exists i \in iter)(b^{(i)} = p \wedge A[a^{(i)}] \geq 0))$$

$$(\forall i)\neg upd_B(i,p) \rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$upd_B(i,p,x) \wedge (\forall j > i)\neg upd_B(j,p) \rightarrow B^{(n)}[p] = x$$

$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)})$$

$$\xrightarrow[\text{Theorem Proving}]{\text{Saturation}} I_1, I_2, I_3, I_4, I_5, \ldots$$

First-Order Saturation Theorem Proving: Proof by Refutation

Given a problem with axioms and assumptions $F_1, \ldots, F_n$ and conjecture G,

1. negate the conjecture;
2. establish unsatisfiability of the set of formulas $F_1, \ldots, F_n, \neg G$.

First-Order Saturation Theorem Proving: Proof by Refutation

Given a problem with axioms and assumptions $F_1, \ldots, F_n$ and conjecture G,

1. negate the conjecture;
2. establish unsatisfiability of the set of formulas $F_1, \ldots, F_n, \neg G$.

Thus, we reduce the theorem proving problem to the problem of checking unsatisfiability.

First-Order Saturation Theorem Proving:
How to Establish Unsatisfiability?

Given a set S_0 of clauses in an inference system $\mathbb{I}$ (e.g. binary resolution or superposition)

First-Order Saturation Theorem Proving: How to Establish Unsatisfiability?

Given a set S_0 of clauses in an inference system $\mathbb{I}$ (e.g. binary resolution or superposition)

Idea:

- Take a set of clauses S (the search space), initially $S = S_0$. Repeatedly apply inferences in $\mathbb{I}$ to clauses in S and add their conclusions to S, unless these conclusions are already in S.
- If, at any stage, we obtain $\bot$, we terminate and report unsatisfiability of S_0.

First-Order Saturation Theorem Proving:
How to Establish Satisfiability?

When can we report satisfiability of S?

First-Order Saturation Theorem Proving:
How to Establish Satisfiability?

When can we report satisfiability of S?

When we build a set S such that any inference applied to clauses in S is already a member of S. Any such set of clauses is called saturated (with respect to $\mathbb{I}$).

First-Order Saturation Theorem Proving:
How to Establish Satisfiability?

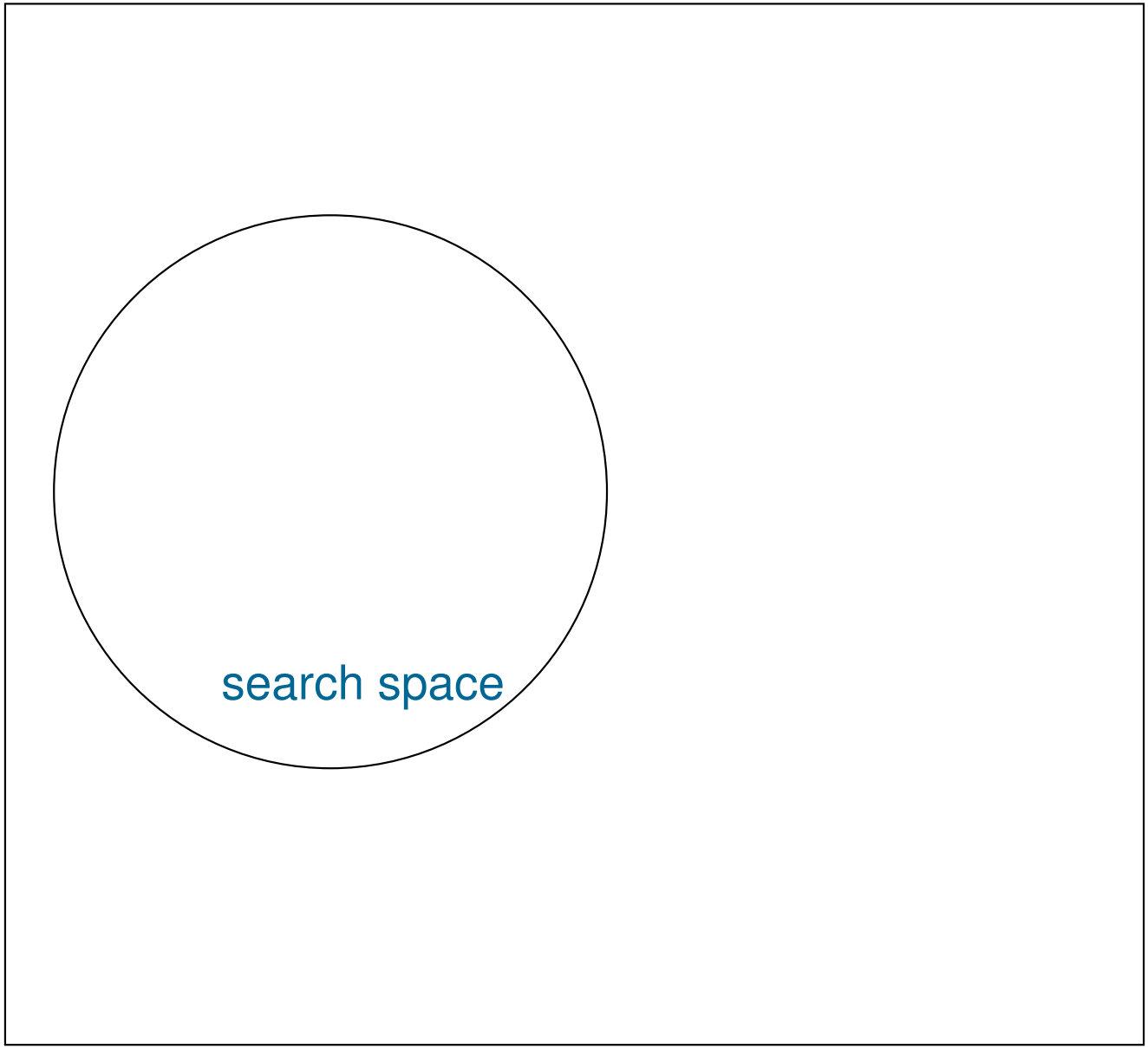
When can we report satisfiability of S?

When we build a set S such that any inference applied to clauses in S is already a member of S. Any such set of clauses is called saturated (with respect to $\mathbb{I}$).

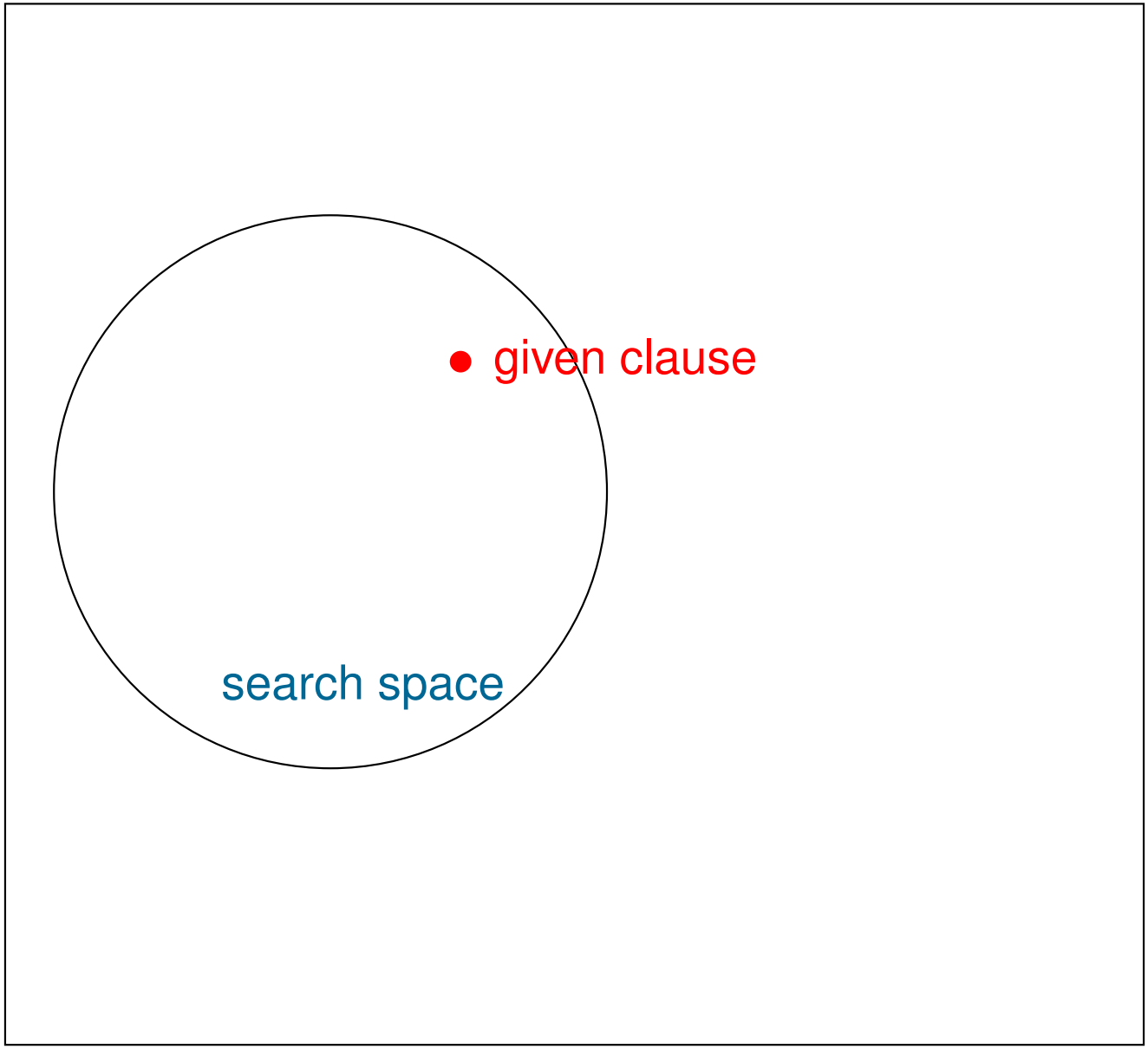
In first-order logic it is often the case that all saturated sets are infinite (due to undecidability), so in practice we can never build a saturated set.

The process of trying to build one is referred to as saturation.

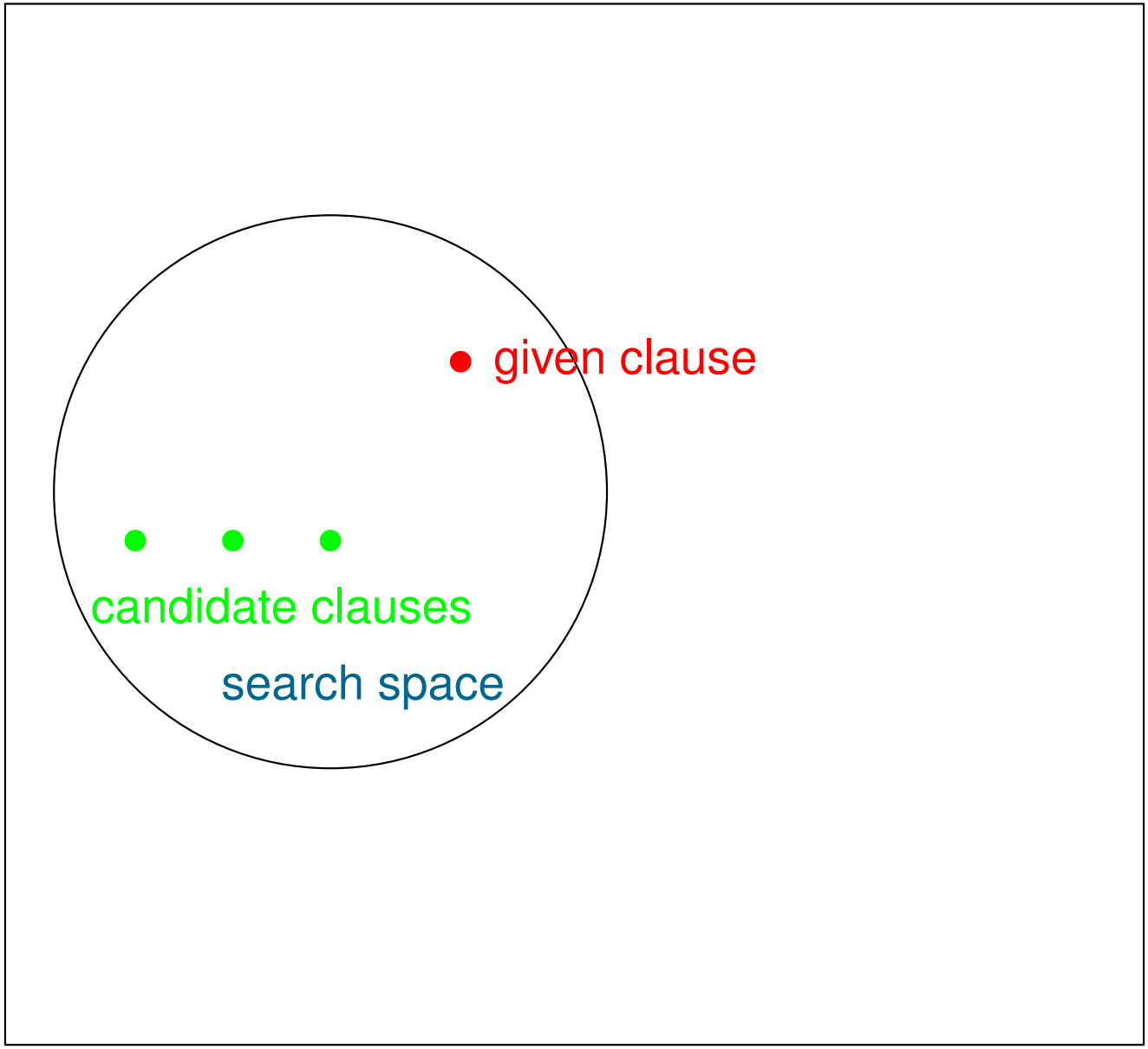
Saturation Algorithms



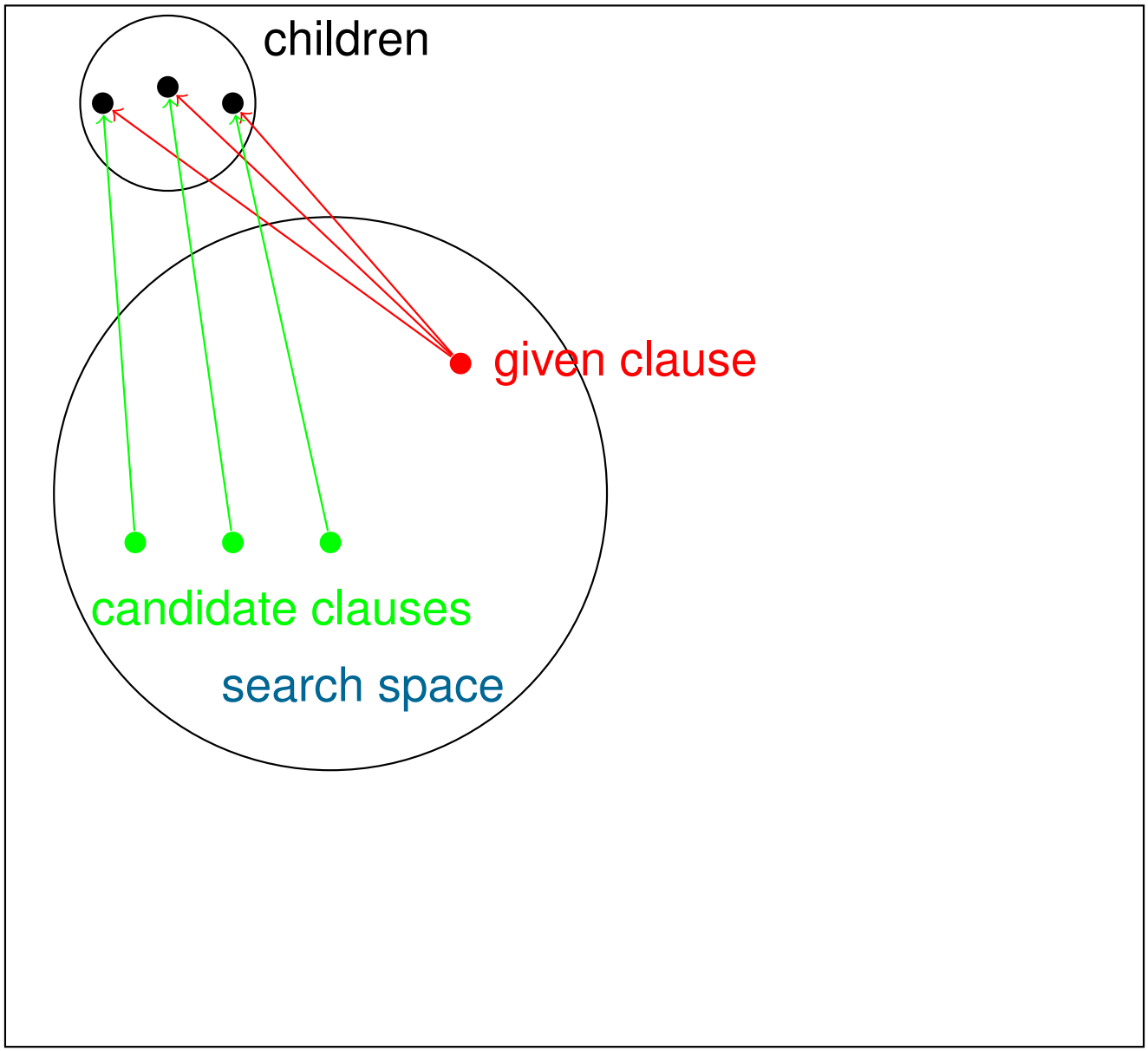
Saturation Algorithms



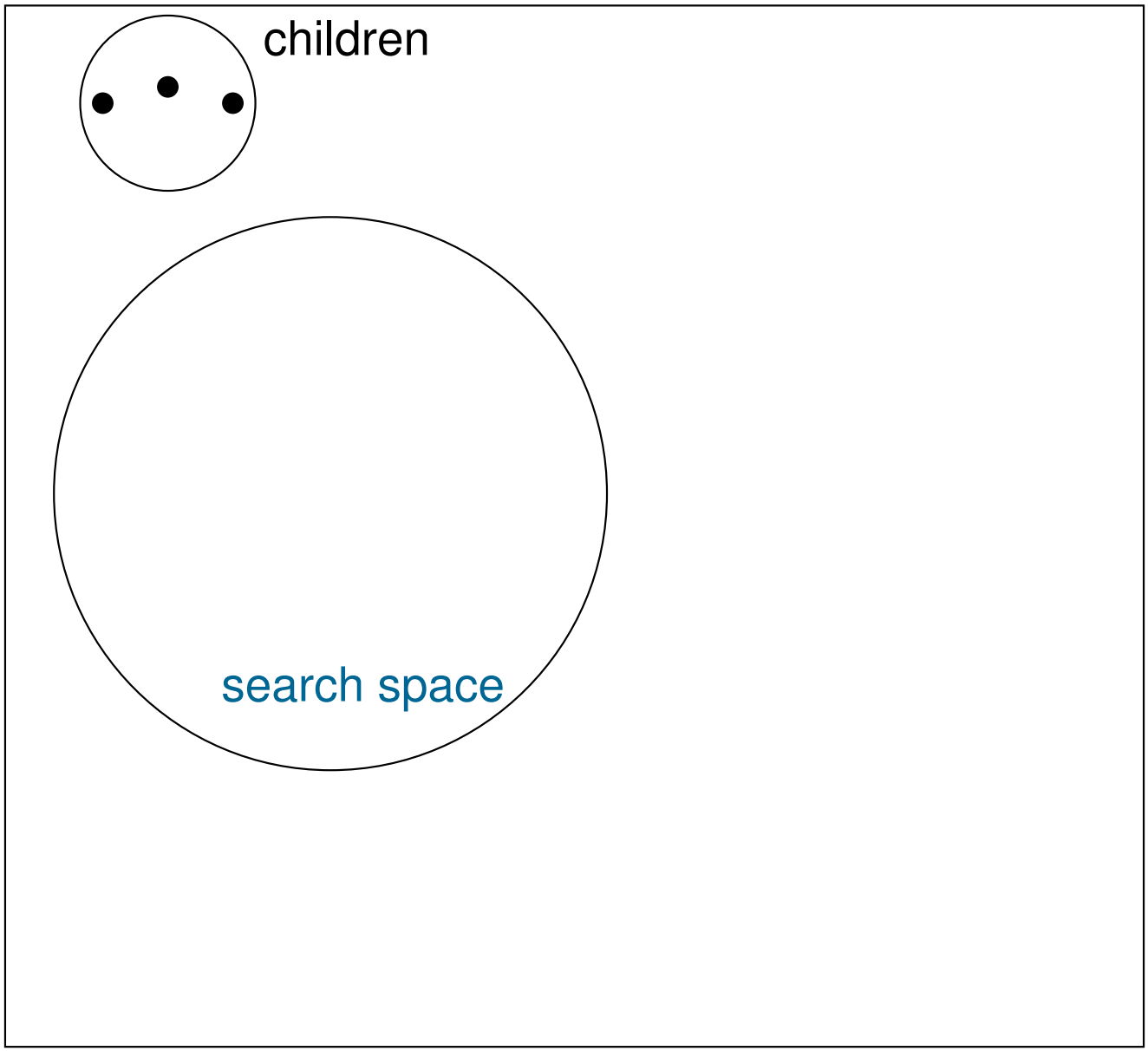
Saturation Algorithms



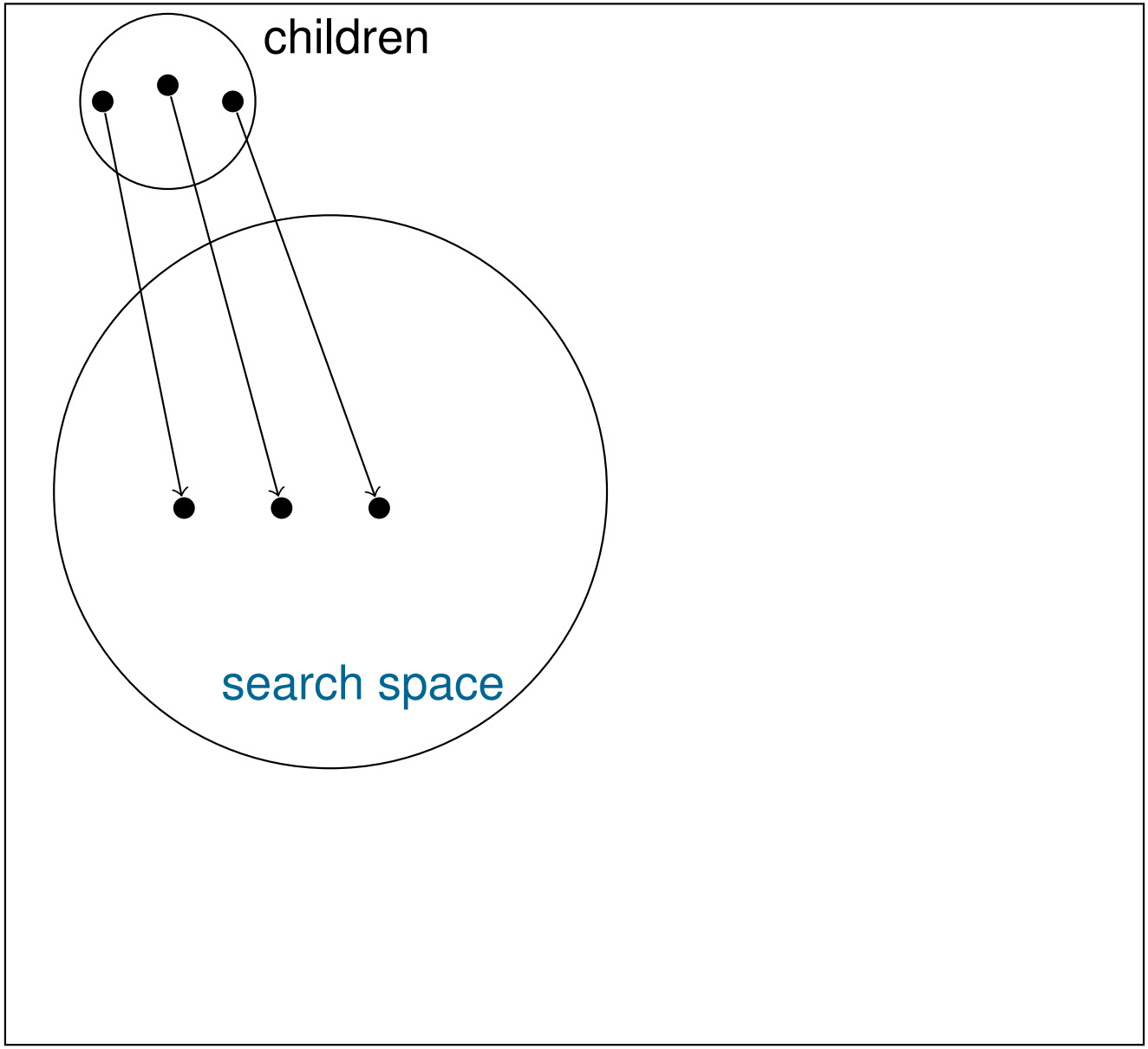
Saturation Algorithms



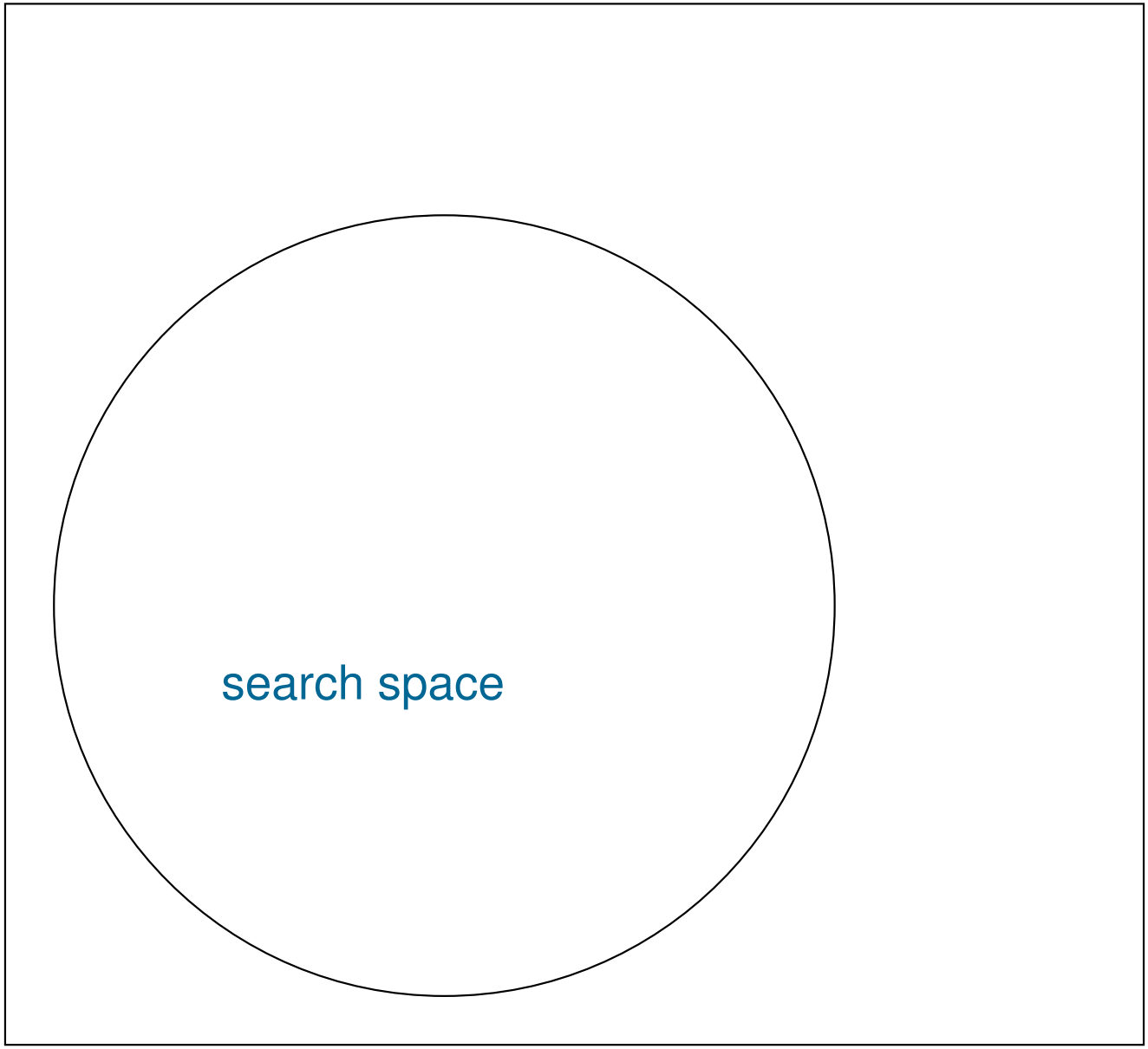
Saturation Algorithms



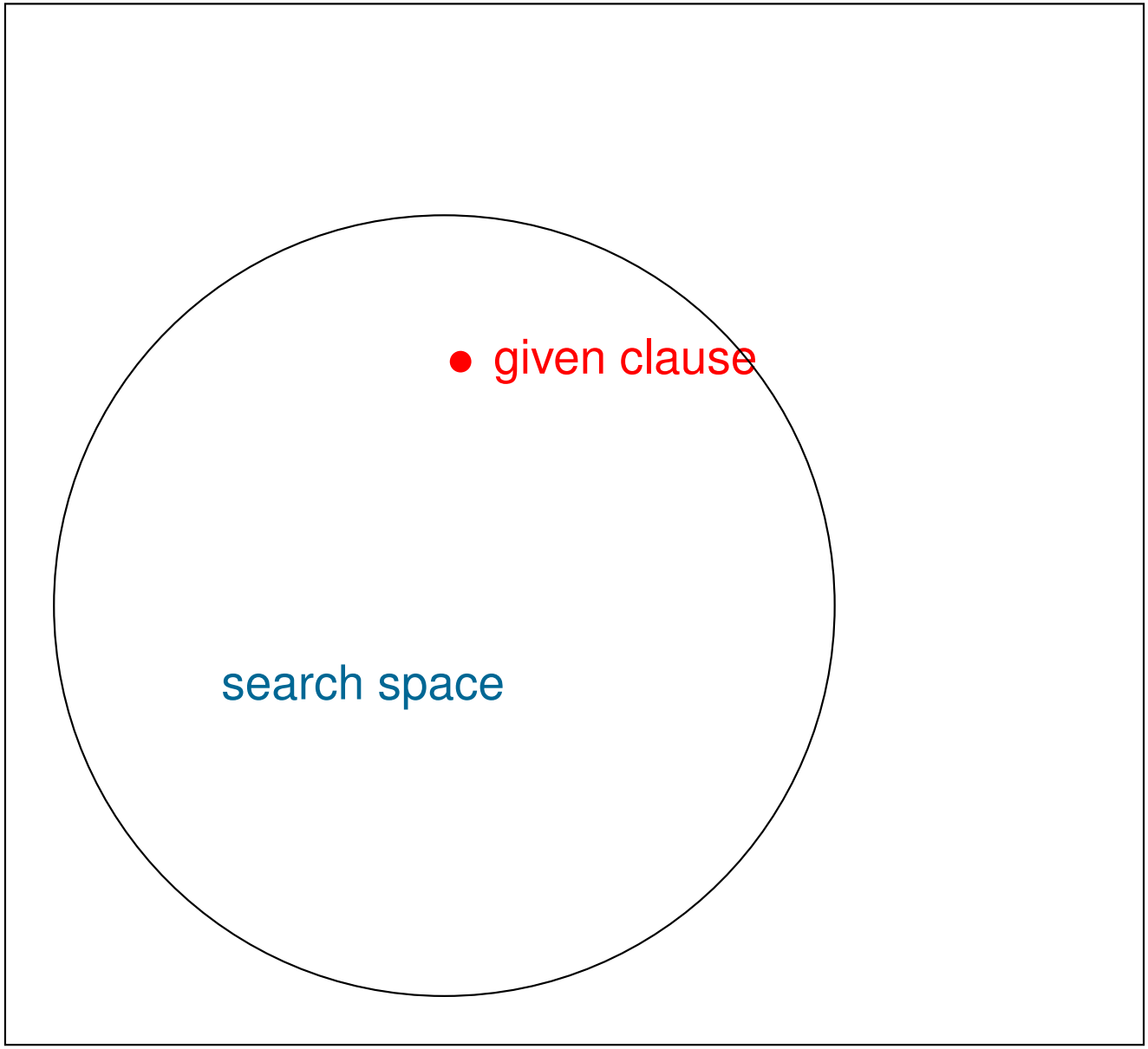
Saturation Algorithms



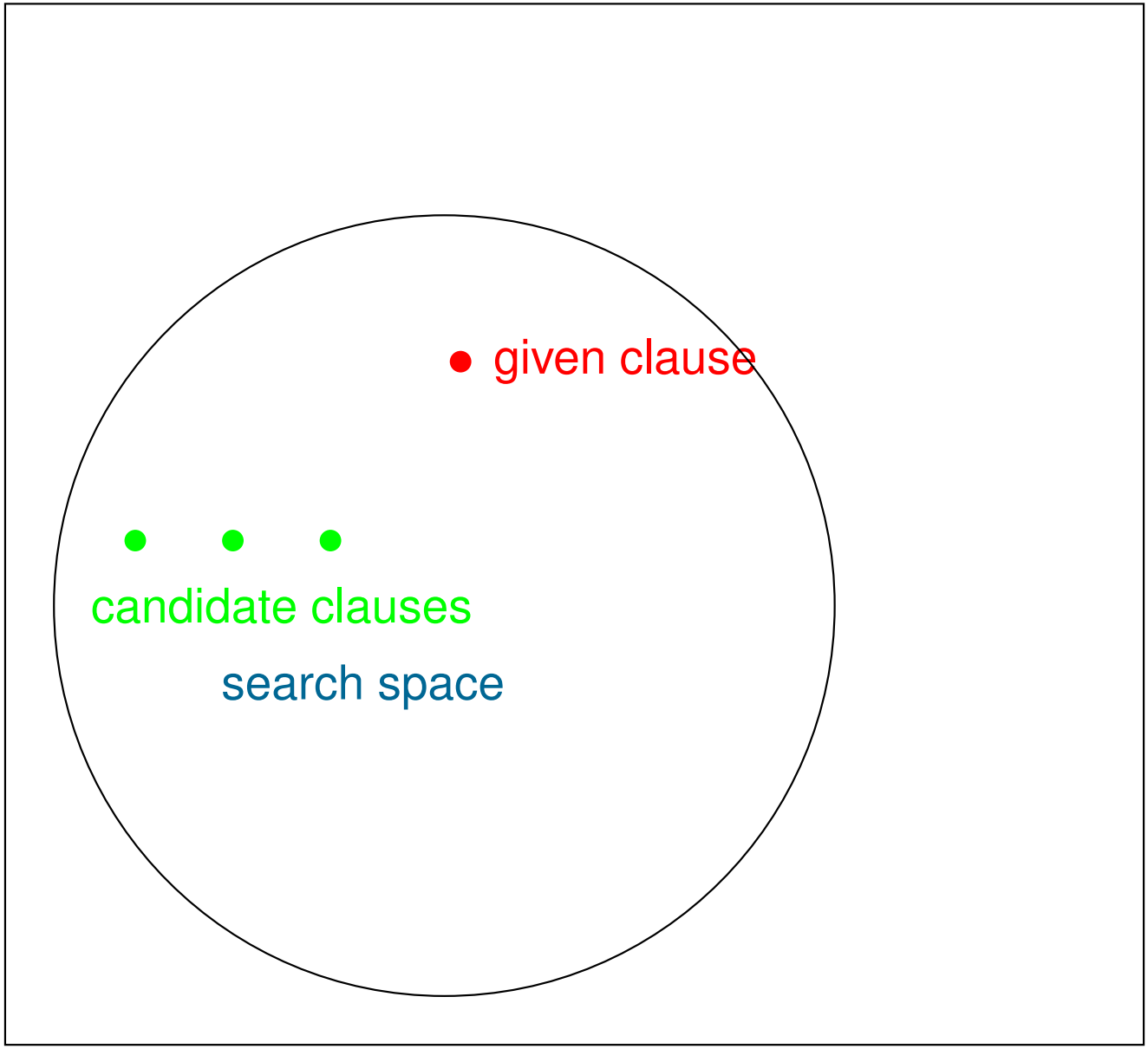
Saturation Algorithms



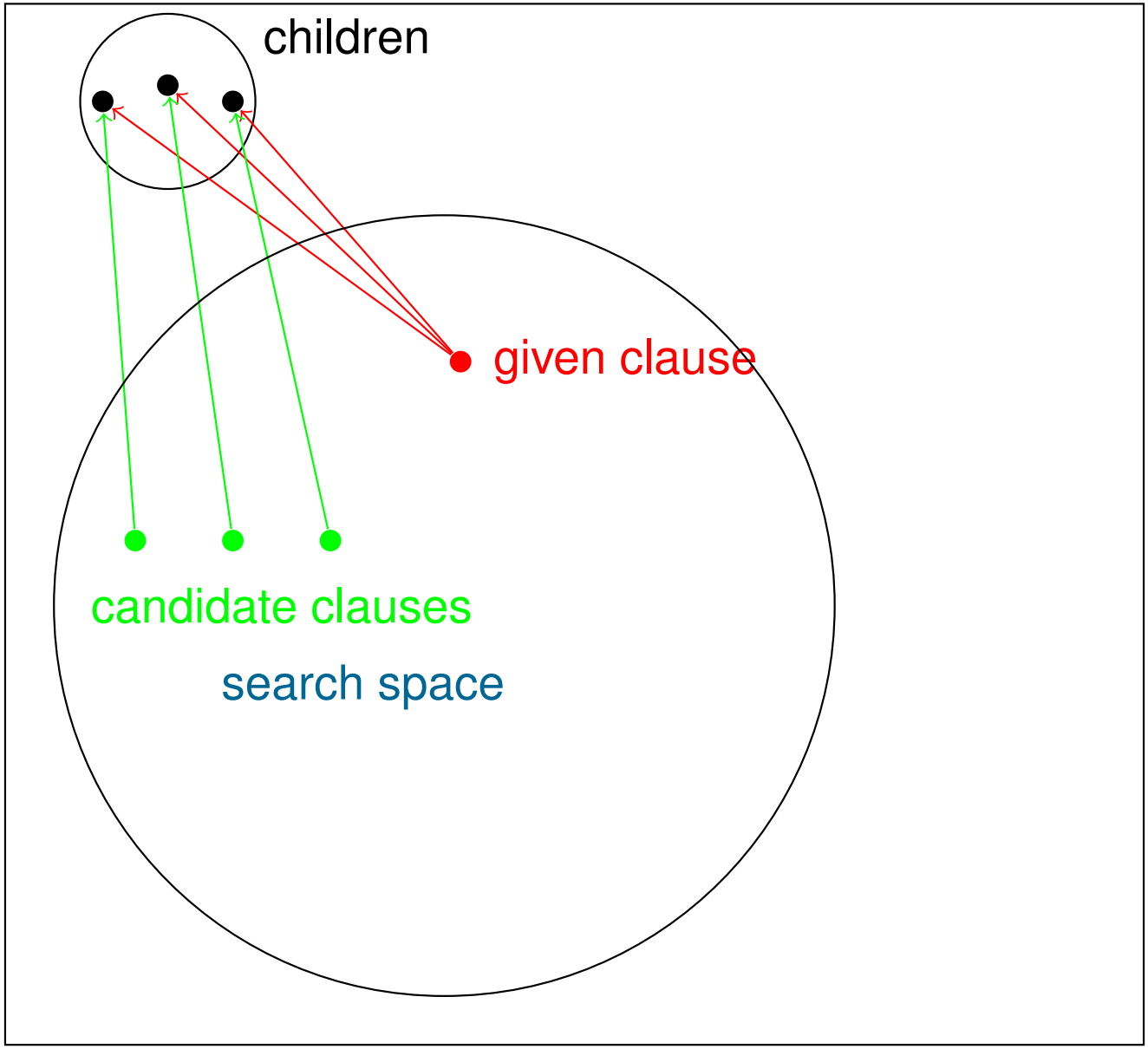
Saturation Algorithms



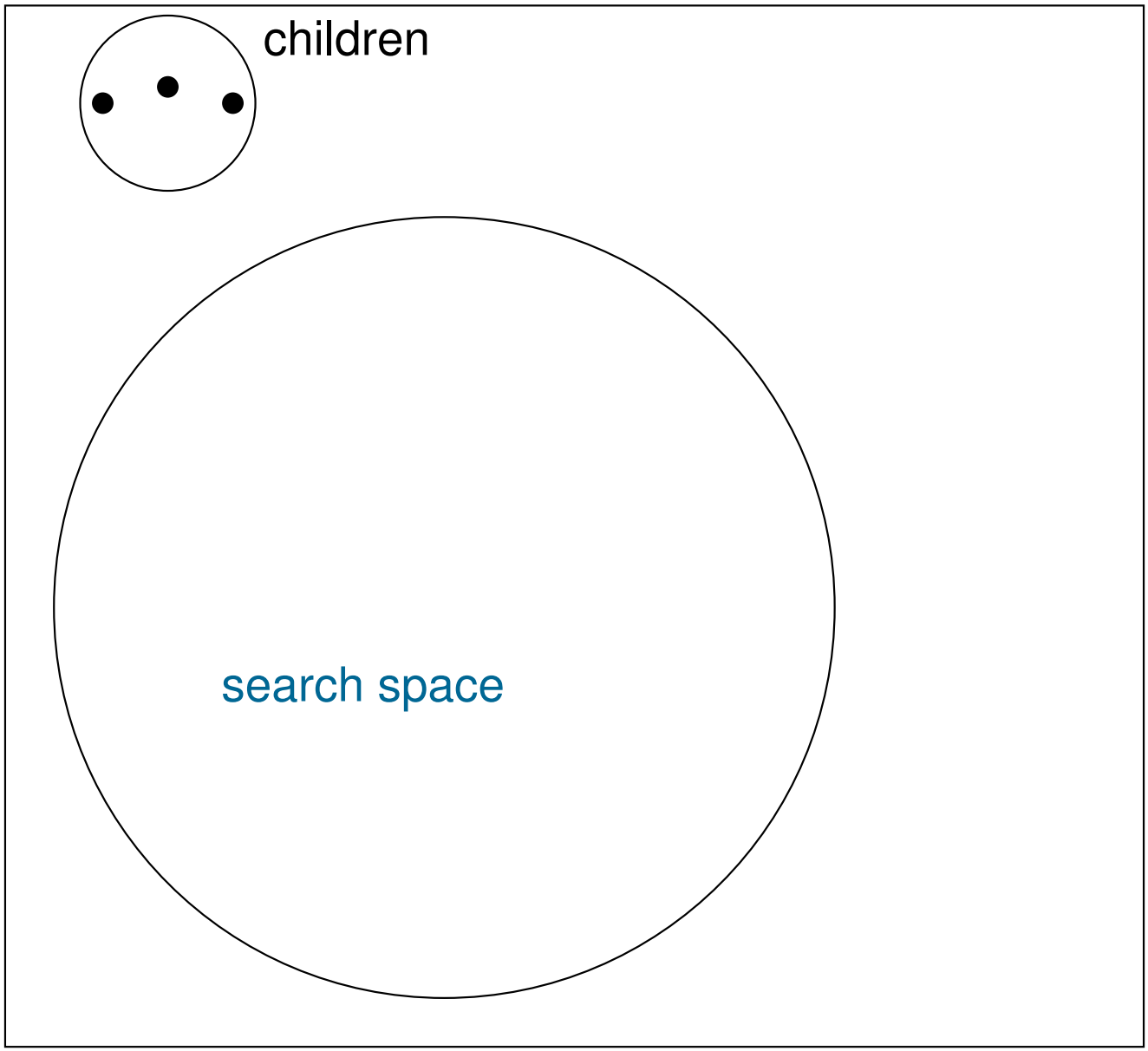
Saturation Algorithms



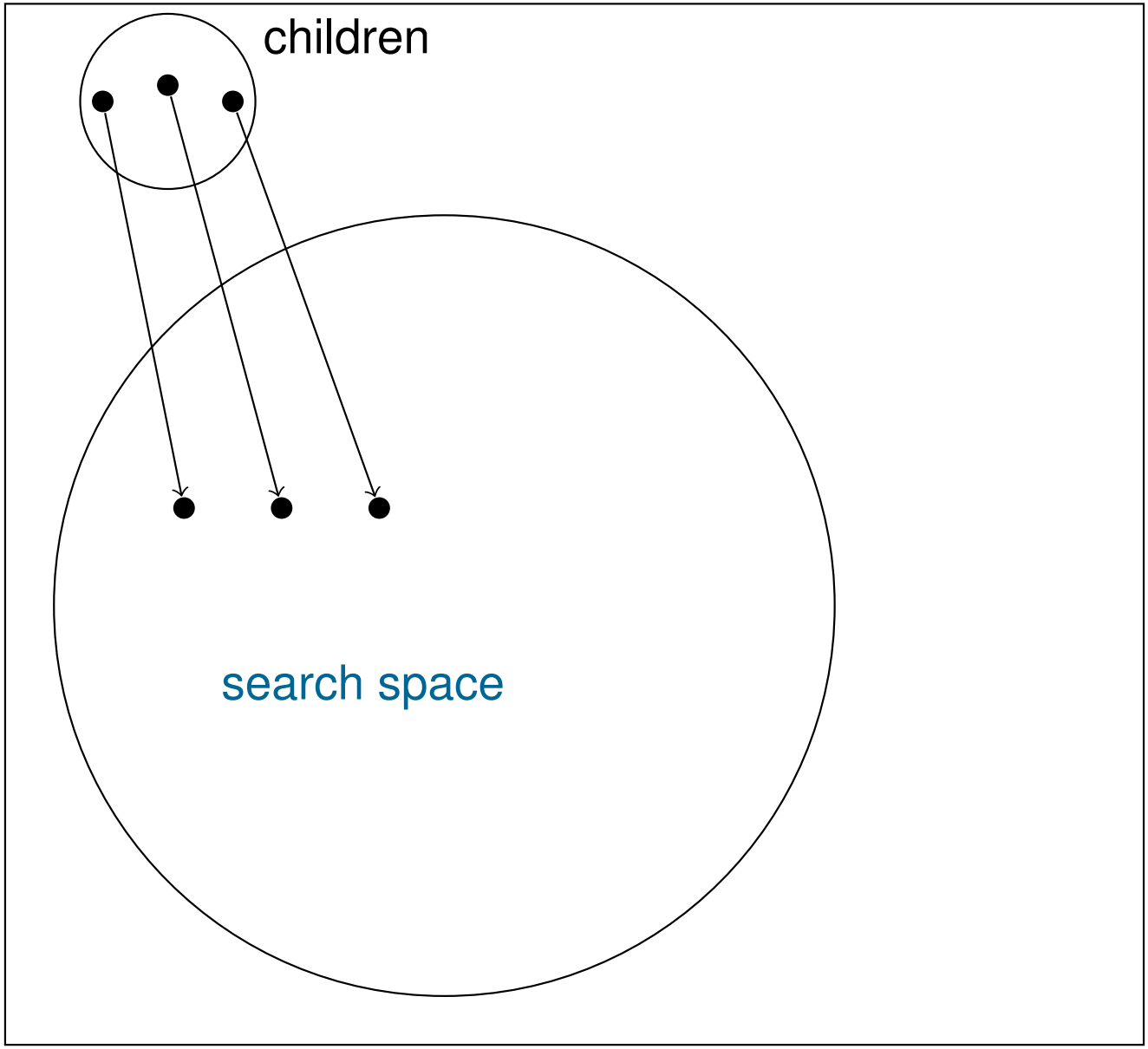
Saturation Algorithms



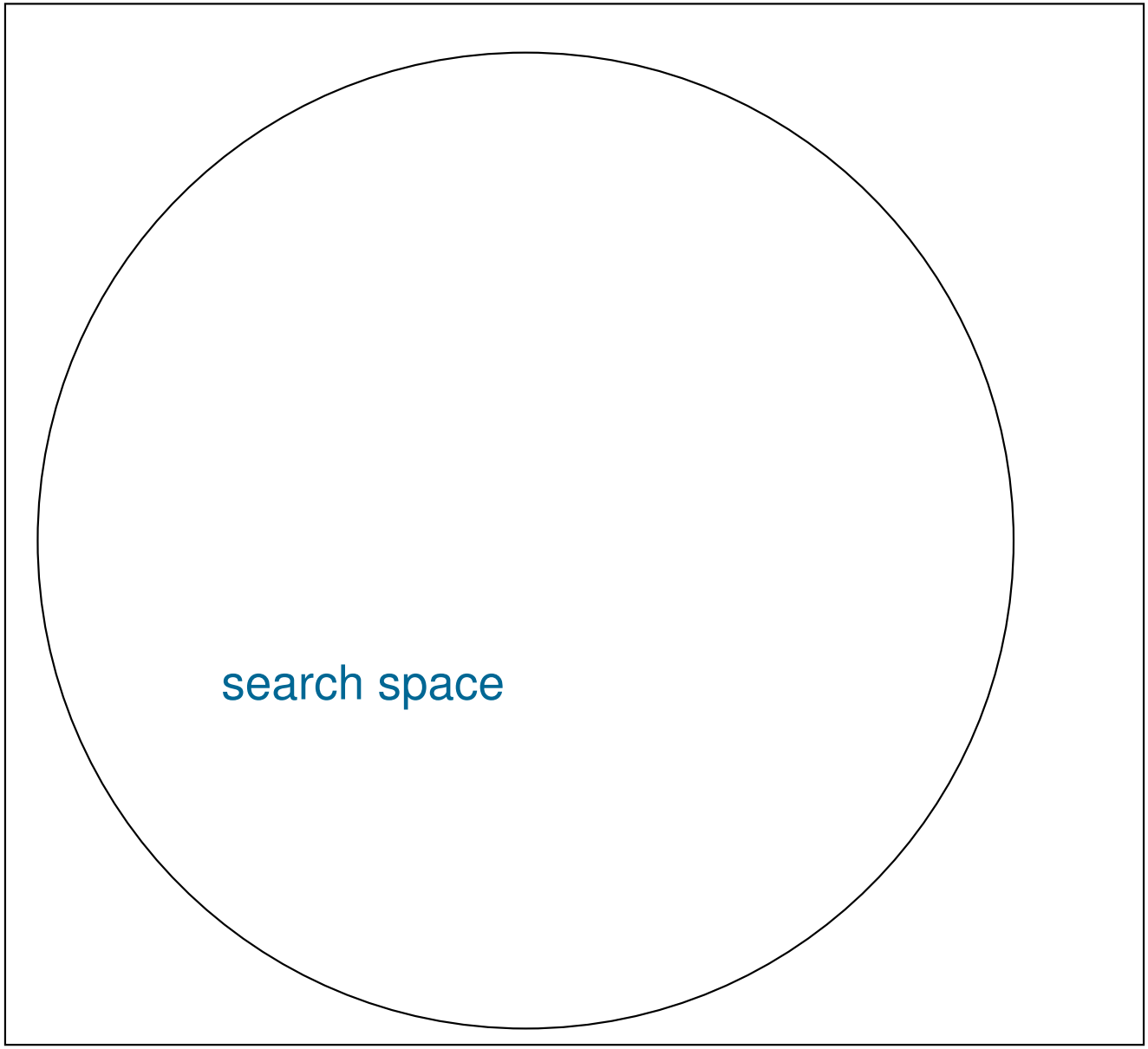
Saturation Algorithms



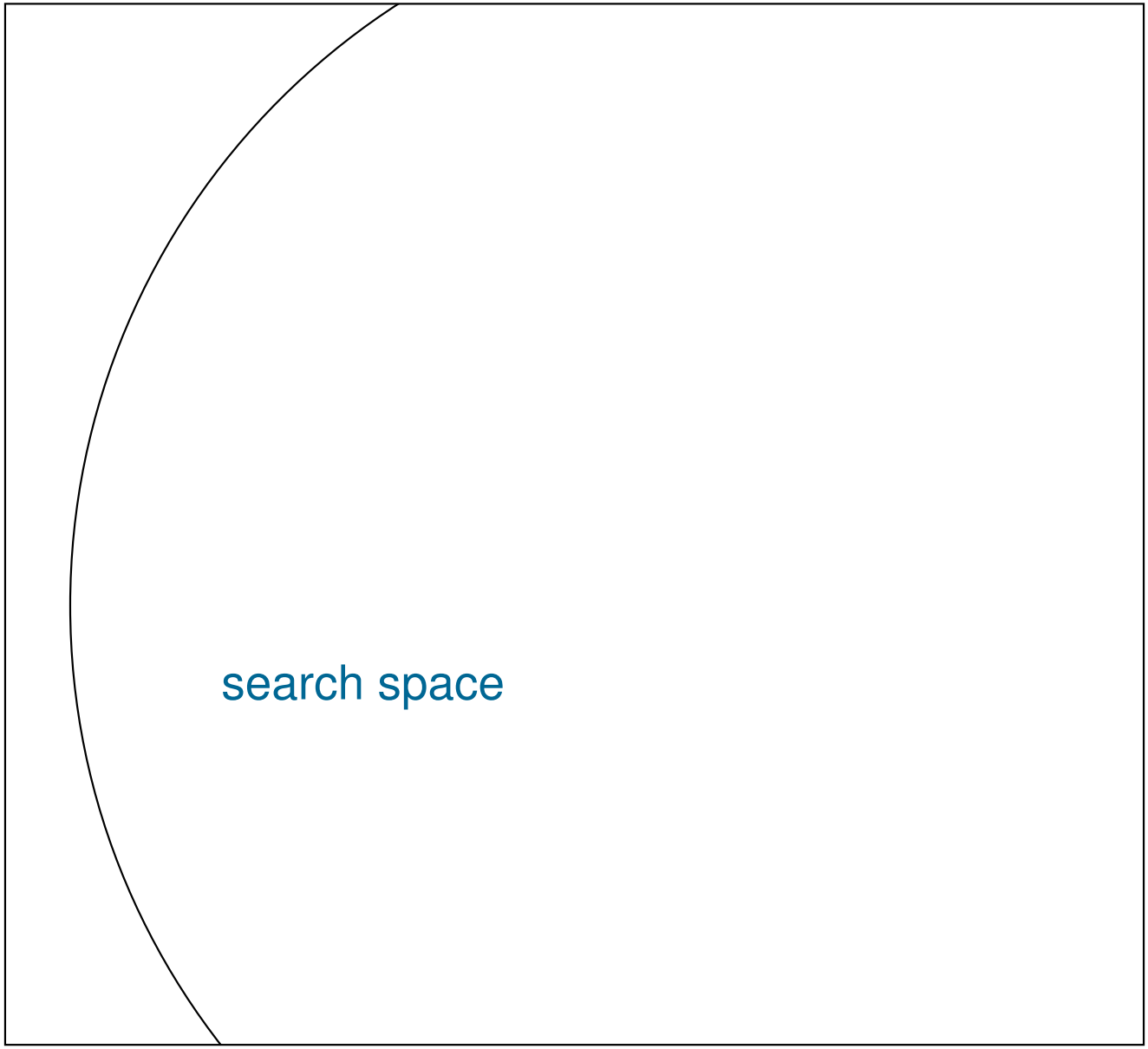
Saturation Algorithms



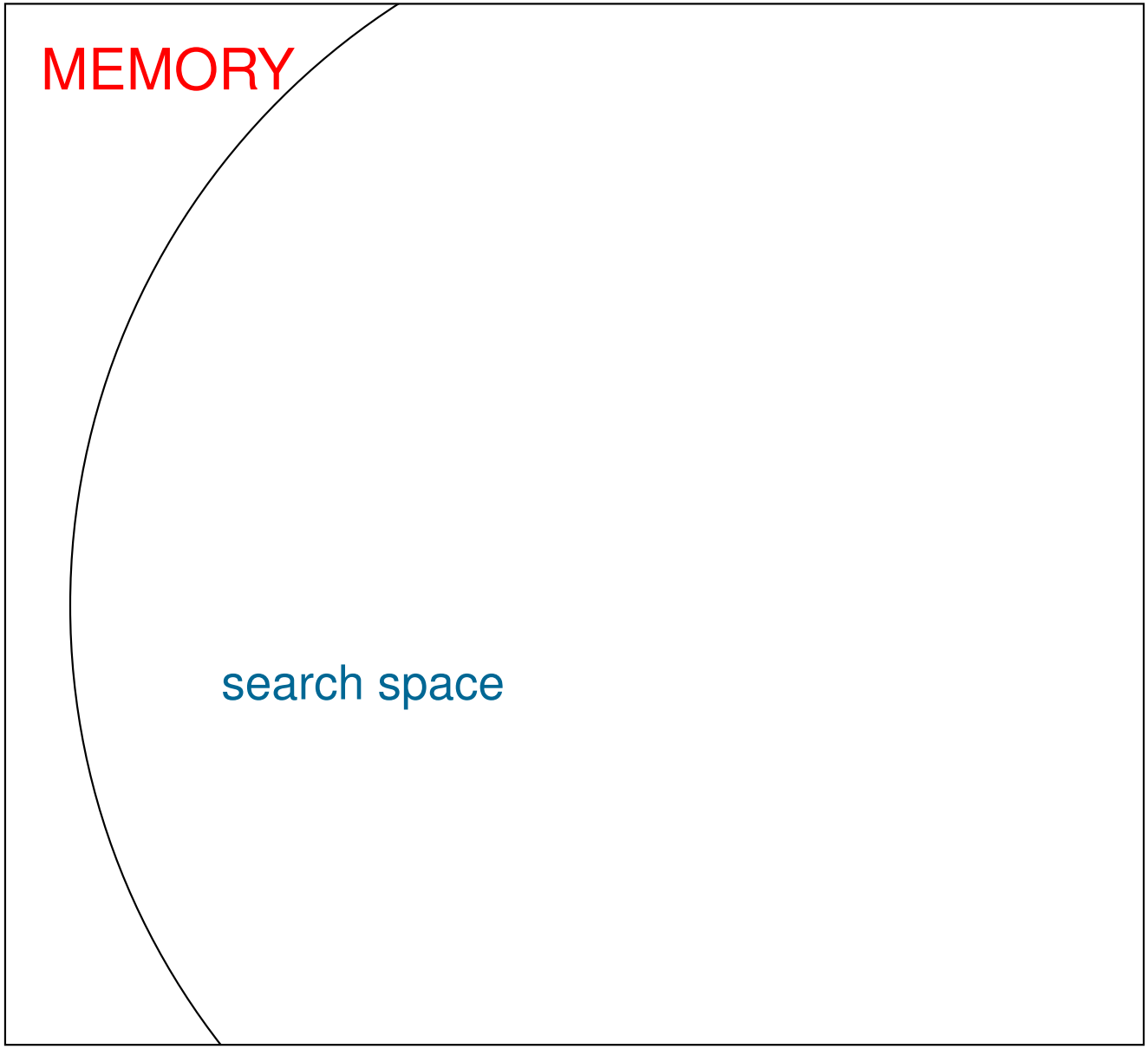
Saturation Algorithms



Saturation Algorithms



Saturation Algorithms



Saturation Algorithm

A saturation algorithm tries to saturate a set of clauses with respect to a given inference system.
In theory there are three possible scenarios:

1. At some moment the empty clause $\bot$ is generated, in this case the input set of clauses is unsatisfiable.
2. Saturation will terminate without ever generating $\bot$, in this case the input set of clauses in satisfiable.
3. Saturation will run forever, but without generating $\bot$. In this case the input set of clauses is satisfiable.

Saturation Algorithm in Practice

In practice there are three possible scenarios:

1. At some moment the empty clause $\bot$ is generated, in this case the input set of clauses is unsatisfiable.
2. Saturation will terminate without ever generating $\bot$, in this case the input set of clauses in satisfiable.
3. Saturation will run until we run out of resources, but without generating $\bot$. In this case it is unknown whether the input set is unsatisfiable.

Symbol Elimination by Saturation Theorem Proving

1. Reasoning in first-order theories

$$
\begin{aligned}
&x \geq y \iff x > y \vee x = y\\
&x > y \rightarrow x \neq y\\
&x \geq y \wedge y \geq z \rightarrow x \geq z\\
&x + 1 > x\\
&x \geq y + 1 \iff x > y
\end{aligned}
$$

2. Procedures for eliminating symbols → USEFUL clauses: Invariants

Symbol Elimination by Saturation Theorem Proving

1. Reasoning in first-order theories

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

- For every loop variable $v \rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (logical variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by the first-order reasoning engine;
- USEFUL clauses:
  - contains only usable symbols;

- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols

Symbol Elimination by Saturation Theorem Proving

1. Reasoning in first-order theories

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

- For every loop variable $v \rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (logical variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by the first-order reasoning engine;
- USEFUL clauses: $x + y = y + x$
  - contains only usable symbols;

- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols

Symbol Elimination by Saturation Theorem Proving

1. Reasoning in first-order theories

$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

- For every loop variable $v \rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (logical variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by the first-order reasoning engine;
- USEFUL clauses: $x + y = y + x$ is not useful
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols

Symbol Elimination by Saturation Theorem Proving

1. Reasoning in first-order theories

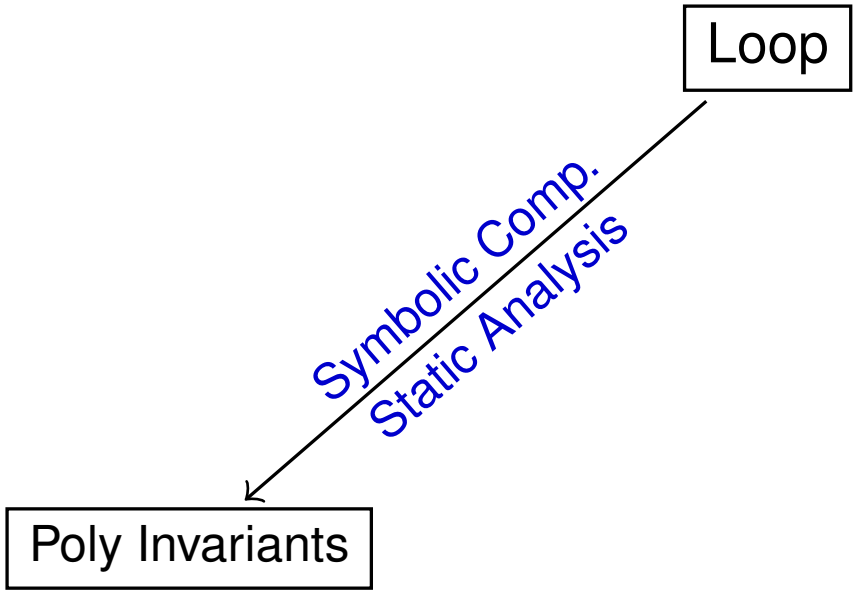
$$x \geq y \iff x > y \vee x = y$$
$$x > y \rightarrow x \neq y$$
$$x \geq y \wedge y \geq z \rightarrow x \geq z$$
$$x + 1 > x$$
$$x \geq y + 1 \iff x > y$$

2. Procedures for eliminating symbols $\rightarrow$ USEFUL clauses: Invariants

- For every loop variable $v \rightarrow$ TARGET SYMBOLS v_0 and v:

$$v^{(0)} = v_0 \qquad \text{and} \qquad v^{(n)} = v$$

- USABLE symbols (logical variables are not symbols):
  - target or interpreted symbols;
  - skolem functions introduced by the first-order reasoning engine;
- USEFUL clauses:
  - contains only usable symbols;
  - contains at least a target symbol or a skolem function;
- Reduction ordering $\succ$: useless symbols $\succ$ usable symbols.



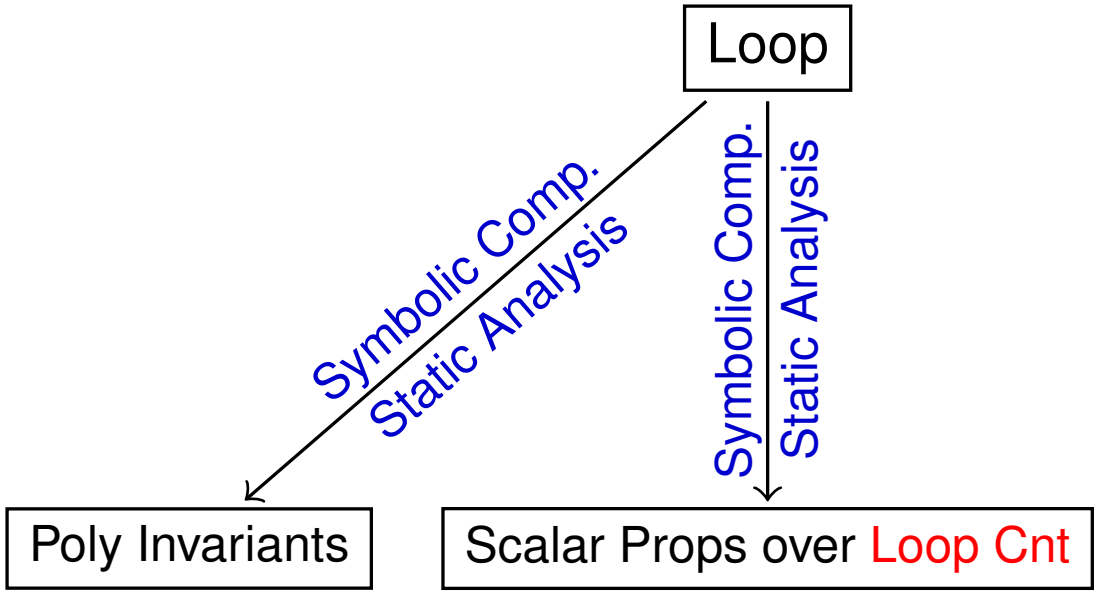

```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

$a = b + c$

$b \geq 0$

$c \geq 0$

$a \geq 0$




```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

$a = b + c$

$b \geq 0$

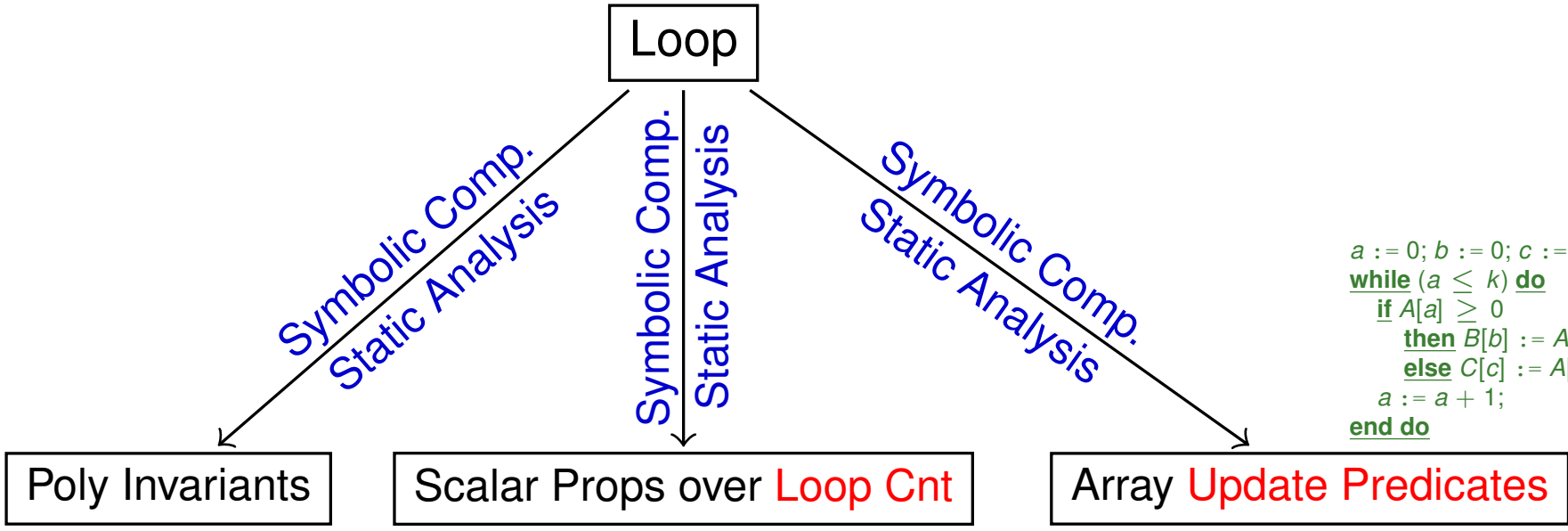
$c \geq 0$

$a \geq 0$

$(\forall i \in \text{iter})(a^{(i)} = a^{(0)} + i)$

$(\forall j, k \in iter)(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$

$(\forall j, k \in iter)(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)$




```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

$a = b + c$

$b \geq 0$

$c \geq 0$

$a \geq 0$

$(\forall i \in \text{iter})(a^{(i)} = a^{(0)} + i)$

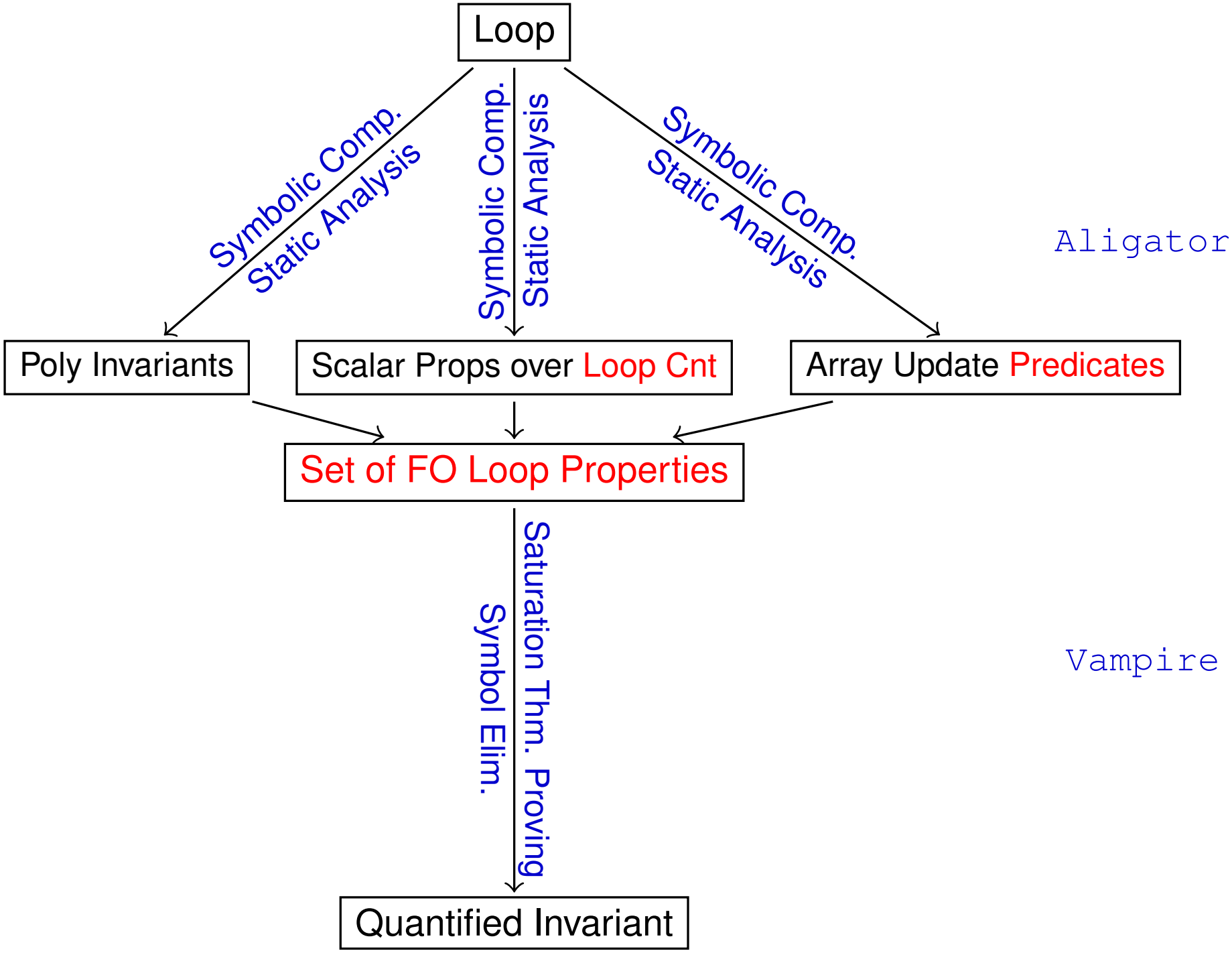
$(\forall j, k \in iter)(k \geq j \Rightarrow b^{(k)} \geq b^{(j)})$

$(\forall j, k \in iter)(k \geq j \Rightarrow b^{(j)} + k \geq b^{(k)} + j)$

$$(\forall i)\neg upd_B(i, p) \Rightarrow B^{(n)}[p] = B^{(0)}[p]$$

$$(\forall v)(b^{(0)} \leq v < b^{(n)} \Rightarrow (\exists i \in iter)(b^{(i)} = v \wedge (A[a^{(i)}] \geq 0))$$

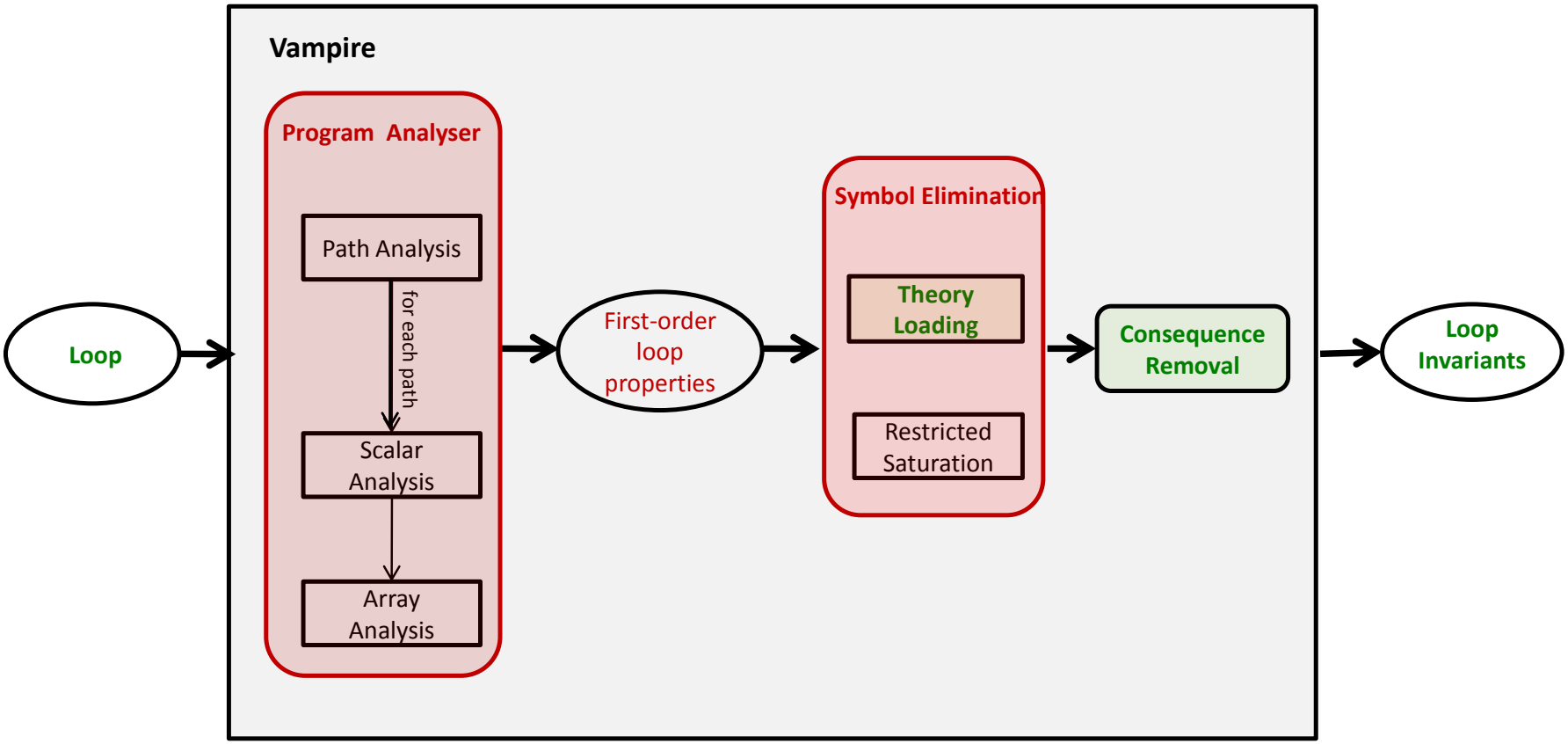
$$(\forall i \in iter)(A[a^{(i)}] \geq 0 \Rightarrow B^{(i+1)}[b^{(i)}] = A[a^{(i)}] \wedge b^{(i+1)} = b^{(i)} + 1 \wedge c^{(i+1)} = c^{(i)} + 1$$


Loop
Symbolic Comp.
Static Analysis
Symbolic Comp.
Static Analysis
Symbolic Comp.
Static Analysis
Aligator
Poly Invariants
Scalar Props over Loop Cnt
Array Update Predicates
Set of FO Loop Properties
Symbol Elim.
Saturation Thm. Proving
Vampire
Quantified Invariant

Outline

Invariant Generation by Symbol Elimination in Vampire

1. Program analysis (new Vampire mode);
2. Theory loading (new Vampire opion);
3. Elimination of “colored” symbols (new Vampire option);
4. Generation of “minimal” set of invariants (new Vampire mode).



1. Sample Output for Program Analysis in Vampire

`vampire --mode program_analysis` partition.c

```
Loops found: 1
Analyzing loop...
---------------------
while (a < m)
{
if (A[a] >= 0)
{
B[b] = A[a]; b = b + 1;
}
else
{
C[c] = A[a]; c = c + 1;
}
a = a + 1;
}
---------------------
Analyzing variables...
---------------------
Variable: A: constant
Variable: C: (updated)
Variable: m: constant
Variable: b: (updated)
Variable: B: (updated)
Variable: c: (updated)
Variable: a: (updated)
Counter: b
Counter: c
Counter: a
```

```
Collecting paths...
---------------------
Path:
false: A[a] >= 0
C[c] = A[a];
c = c + 1;
a = a + 1;
Path:
true: A[a] >= 0
B[b] = A[a];
b = b + 1;
a = a + 1;
Counter a: 1 min, 1 max, 1 gcd
Counter b: 0 min, 1 max, 1 gcd
Counter c: 0 min, 1 max, 1 gcd
...
Collected first-order loop properties...
---------------------
37. iter(X0) <=> (0<= X0 & X0<n) [program analysis]
...
7. ![X1,X0,X3]:(X1>X0 & c(X1)>X3 & X3>c(X0)) =>
   ?[X2]:(c(X2)=X3 & X2>X0 & X1>X2) [program analysis]
6. ![X0]:c(X0)>=c0 (0:4) [program analysis]
5. ![X0]:c(X0)<=c0+X0 (0:6) [program analysis]
4. ![X1,X0,X3]:(X1>X0 & b(X1)>X3 & X3>b(X0))
   => ?[X2]:(b(X2)=X3 & X2>X0 & X1>X2) [program
analysis]
3. ![X0]:b(X0)>=b0 (0:4) [program analysis]
2. ![X0]:b(X0)<=b0+X0 (0:6) [program analysis]
1. ![X0]:a(X0)=a0+X0 (0:6) [program analysis]
```

Figure : Partial output of Vampire's program analyser on the `Partition` program.

1. Program Analysis in Vampire: If-then-else and Let-in

A partial correctness statement:

```
{∀X(p(X) => X ≥ 0)}
{∀X(q(X) > 0)}
{p(a)}
if (r(a)) {
  a := a+1
}
else {
  a := a + q(a).
}
{a > 0}
```

1. Program Analysis in Vampire: If-then-else and Let-in

A partial correctness statement:

```
{∀X(p(X) => X ≥ 0)}
{∀X(q(X) > 0)}
{p(a)}
if (r(a)) {
  a := a+1
}
else {
  a := a + q(a).
}
{a > 0}
```

The next state function for `a`:

```
a' =
  if r(a)
  then let a=a+1 in a
  else let a=a+q(a) in a
```

1. Program Analysis in Vampire: If-then-else and Let-in

A partial correctness statement:

```
{∀X(p(X) => X ≥ 0)}
{∀X(q(X) > 0)}
{p(a)}
if (r(a)) {
  a := a+1
}
else {
  a := a + q(a).
}
{a > 0}
```

The next state function for `a`:

```
a' =
  if r(a)
  then let a=a+1 in a
  else let a=a+q(a) in a
```

In Vampire:

```
tff(1,type,p : $int > $o).
tff(2,type,q : $int > $int).
tff(3,type,r : $int > $o).
tff(4,type,a : $int).

tff(5,hypothesis,! [X:$int] :
  (p(X) => $greatereq(X,0))).
tff(6,hypothesis,! [X:$int] :
  ($greatereq(q(X),0))).
tff(7,hypothesis,p(a)).

tff(8,hypothesis,
  a0 = $ite_t(r(a),
    $let_tt(a,$sum(a,1),a),
    $let_tt(a,$sum(a,q(a)),a)
)).

tff(9,conjecture,$greater(a0,0)).
```

2. Theory Loading in Vampire

We use incomplete but sound theory axiomatisation.

Example: Integers in Vampire

- 0, 1, 2, etc;
- Integer predicates/funcions:
  - addition;
  - subtraction;
  - multiplication;
  - successor;
  - division;
  - inequality relations;

2. Theory Loading in Vampire: Sorts in TPTP

```
tff(boolean_type,type,b: $tType).    % b is a sort
tff(s_is_a_type,type,s: $tType).     % s is a sort

tff(t_has_type_b,type,t : b).    % t has sort b
tff(f_has_type_b,type,f : b).    % f has sort b

tff(1,axiom,t != f & ! [X:b] : X = t | X = f).
tff(1,axiom,? [X:s,Y:s,Z:s] : (X != Y & X != Z & Y != Z)).
```

```
vampire --splitting off
  --saturation_algorithm inst_gen sort2.tptp
```

2. Theory Loading in Vampire: Pre-existing sorts

- `$i`: sort of individuals. If is the default sort: if a symbol is not declared, it has this sort.
- `$o`: sort of booleans.
- `$int`: sort of integers.
- `$rat`: sort of rationals.
- `$real`: sort of reals.

2. Theory Loading in Vampire: Integers

One can use concrete integers and some interpreted functions on them.

```
fof(1,conjecture,$sum(2,2)=4).
```

```
vampire --inequality_splitting 0 int1.tptp
```

2. Theory Loading in Vampire: Interpreted Fct/Predicates on Int

Functions:

- `$sum`: addition ($x + y$)
- `$product`: multiplication ($x \cdot y$)
- `$difference`: difference ($x - y$)
- `$uminus`: unary minus ($-x$)
- `$to_rat`: conversion to rationals.
- `$to_real`: conversion to reals.

Predicates:

- `$less`: less than ($x < y$)
- `$lesseq`: less than or equal to ($x \leq y$)
- `$greater`: greater than ($x > y$)
- `$greatereq`: greater than or equal to ($x \geq y$)

2. Theory Loading in Vampire: How Vampire Proves in Arithmetic

- adding theory axioms;
- evaluating expressions, when possible;
- (future) SMT solving.

2. Theory Loading in Vampire: How Vampire Proves in Arithmetic

- adding theory axioms;
- evaluating expressions, when possible;
- (future) SMT solving.

Example:

$$(x + y) + z = x + (z + y).$$

```
fof(1,conjecture,
  ! [X:$int,Y:$int,Z:$int] :
    $sum($sum(X,Y),Z)=$sum(X,$sum(Z,Y))).
```

```
vampire --inequality_splitting 0 int2.tptp
```

2. Theory Loading in Vampire: How Vampire Proves in Arithmetic

- adding theory axioms;
- evaluating expressions, when possible;
- (future) SMT solving.

Example:

$$(x + y) + z = x + (z + y).$$

```
fof(1,conjecture,
  ! [X:$int,Y:$int,Z:$int] :
    $sum($sum(X,Y),Z)=$sum(X,$sum(Z,Y))).
```

```
vampire --inequality_splitting 0 int2.tptp
```

- You can add your own axioms;
- you can replace Vampire axioms by your own: use

  `--theory_axioms off`

3. Elimination of Colored Symbols in Vampire

vampire(option,show_symbol_elimination,on).

```
vampire(option,time_limit,1).

...
tff(b_type,type,a:$int).
tff(b_fcttype,type,a:$int>$int).
tff(bb_type,type,bb:$int>$int).
tff(bb_fct2type,type,bb:($int*$int)>$int).
tff(iter_fcttype,type,iter:$int>$o).
tff(upd2_type,type,updbb:($int*$int)>$o).
tff(upd3_type,type,updbb:($int*$int*$int)>$o).
...

vampire(symbol,function,n,0,left).
vampire(symbol,function,a,1,left).
vampire(symbol,function,b,1,left).
vampire(symbol,function,c,1,left).
vampire(symbol,function,bb,2,left).
vampire(symbol,function,cc,2,left).
vampire(symbol,predicate,updB,2,left).
vampire(symbol,predicate,updB,3,left).
vampire(symbol,predicate,updC,2,left).
vampire(symbol,predicate,updC,3,left).
vampire(symbol,predicate,iter,1,left).

vampire(symbol,function,a,0,skip).
vampire(symbol,function,b,0,skip).
vampire(symbol,function,c,0,skip).
vampire(symbol,function,m,0,left).
vampire(symbol,function,aa,1,skip).
vampire(symbol,function,bb0,2,skip).
vampire(symbol,function,bb0,1,skip).
vampire(symbol,function,cc0,2,skip).
vampire(symbol,function,cc0,1,skip).
```

Figure : Partial input for symbol elimination in Vampire.

./vampire array_partition.tptp

4. Generation of Minimal Set of Invariants in Vampire

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

4. Generation of Minimal Set of Invariants in Vampire

```
vampire --mode consequence_elimination
```

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

4. Generation of Minimal Set of Invariants in Vampire

```
vampire --mode consequence_elimination
```

Set of invariants: S

Minimal set S' of invariants with $S' \subset S$:

Remove $C \in S$ iff $S \setminus \{C\} \Rightarrow C$

Compute $S' \subset S$

Run Vampire on S within, e.g., 20s time limit

Experiments:

- consequence elimination ran in conjunction with 4 combination of strategies
- eliminated $\sim$ 80% invariants

Loop	♯ SEI	♯ Min SEI	Inv of interest	Generated invariants implying Inv

`Copy`
$a = 0;$
while $(a < m)$ **do**
$B[a] = A[a]; a = a + 1$
end do
	24	5	$\forall x: \quad 0 \leq x < a \rightarrow B[x] = A[x]$	`inv8`:
				$\forall x_0, x_1: \quad A[x_0] = B[x_1] \vee x0 \neq x_1 \vee \neg a > x_0 \vee \neg x_0 \geq 0$

`Find`
$a = 0; spot = m$
while $(a < m)$ **do**
if $(spot = m \&\& A[a] \neq 0)$
then $spot = a$
end if;
$B[a] = (A[a] \neq 0);$
$a = a + 1$
end do
	151	13	$spot = m \vee A[spot] \neq 0$	`inv3`:
				$a \geq spot$
				`inv39`:
				$spot = sk_1 \vee a = spot$
				`inv25`:
				$0 \geq sk_1 \vee a = spot$
				`inv5`:
				$\forall x_1: \quad \neg a > x_1 \vee \neg x_1 \geq 0 \vee a = spot \vee A(spot) \neq 0$

`Partition`
$a = 0; b = 0; c = 0;$
while $(a < m)$ **do**
if $(A[a] >= 0)$
then $B[b] = A[a]; b = b + 1$
else $C[c] = A[a]; c = c + 1$
end if;
$a = a + 1$
end do
	166	38	$\forall x: \quad 0 \leq x < b \rightarrow B[x] \geq 0 \wedge$
			$\exists y: \quad B[x] = A[y]$
				`inv1`:
				$\forall x_0: \quad A(sk_2(x_0)) \geq 0 \vee \neg b > x_0 \vee \neg x_0 \geq 0$
				`inv81`:
				$\forall x_0: \quad \neg b > x_0 \vee \neg x_0 \geq 0 \vee A(sk_2(x_0)) = B(x_0)$

`Partition_Init`
$a = 0; c = 0;$
while $(a < m)$ **do**
if $(A[a] == B[a])$
then $C[c] = a; c = c + 1$
end if;
$a = a + 1$
end do
	168	24	$\forall x: \quad 0 \leq x < c \rightarrow A[C[x]] = B[C[x]]$	`inv0`:
				$\forall x_0: \quad A(sk_1(x_0)) = B(sk_1(x_0)) \vee \neg c > x_0 \vee \neg x_0 \geq 0$
				`inv30`:
				$\forall x_0, x_1, x_2: \quad sk_1(x_0) \neq x_1 \vee x_0 \neq x_2 \vee \neg c > x_0 \vee \neg x_0 \geq 0 \vee C(x_2) = x_1$

Table : Vampire with 1 second time limit.

Outline

Conclusions: Quantified Invariant Generation and Symbol Elimination

Given a loop:

1. Express loop properties in a language containing extra symbols (loop counter, predicates expressing array updates, etc.);
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover for eliminating extra symbols;
4. Every derived formula in the language of the loop is a loop invariant;
5. Invariants are **consequences of symbol-eliminating inferences (SEI)**.

SEI: premise contains extra symbols, conclusion is in the loop language.

End of Session 3

Slides for session 3 ended here . . .