Automated Reasoning for Systems Engineering

Laura Kovács

Vienna University of Technology

Future and Our Motivation

1. Automated reasoning, in particular theorem proving will remain central in software verification and program analysis.

   The role of theorem proving in these areas will be growing.

2. Theorem provers will be used by a large number of users who do not understand theorem proving and by users with very elementary knowledge of logic.

3. Reasoning with both quantifiers and theories will remain the main challenge in practical applications of theorem proving (at least) for the next decade.

4. Theorem provers will be used in reasoning with very large theories. These theories will appear in knowledge mining and natural language processing.

Future and Our Motivation

1. Automated reasoning, in particular theorem proving will remain central in software verification and program analysis.

   The role of theorem proving in these areas will be growing.

2. Theorem provers will be used by a large number of users who do not understand theorem proving and by users with very elementary knowledge of logic.

3. Reasoning with both quantifiers and theories will remain the main challenge in practical applications of theorem proving (at least) for the next decade.

4. Theorem provers will be used in reasoning with very large theories. These theories will appear in knowledge mining and natural language processing.

Future and Our Motivation

1. Automated reasoning, in particular theorem proving will remain central in software verification and program analysis.

   The role of theorem proving in these areas will be growing.

2. Theorem provers will be used by a large number of users who do not understand theorem proving and by users with very elementary knowledge of logic.

3. Reasoning with both quantifiers and theories will remain the main challenge in practical applications of theorem proving (at least) for the next decade.

4. Theorem provers will be used in reasoning with very large theories. These theories will appear in knowledge mining and natural language processing.

Future and Our Motivation

1. Automated reasoning, in particular theorem proving will remain central in software verification and program analysis.

   The role of theorem proving in these areas will be growing.

2. Theorem provers will be used by a large number of users who do not understand theorem proving and by users with very elementary knowledge of logic.

3. Reasoning with both quantifiers and theories will remain the main challenge in practical applications of theorem proving (at least) for the next decade.

4. Theorem provers will be used in reasoning with very large theories. These theories will appear in knowledge mining and natural language processing.

Outline

First-Order Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

First-Order Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.
More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

First-Order Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.
More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."
What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Formulation in First-Order Logic

Axioms (of group theory):	$\forall x(1 \cdot x = x)$
	$\forall x(x^{-1} \cdot x = 1)$
	$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$
Assumptions:	$\forall x(x \cdot x = 1)$
Conjecture:	$\forall x \forall y(x \cdot y = y \cdot x)$

In the TPTP Syntax

The TPTP library (Thousands of Problems for Theorem Provers), `http://www.tptp.org` contains a large collection of first-order problems. For representing these problems it uses the TPTP syntax, which is understood by all modern theorem provers, including our Vampire prover.

In the TPTP Syntax

The TPTP library (Thousands of Problems for Theorem Provers), `http://www.tptp.org` contains a large collection of first-order problems. For representing these problems it uses the TPTP syntax, which is understood by all modern theorem provers, including our Vampire prover.

First-Order Logic (FOL)	TPTP
$\bot, \top$	`$false, $true`
$\neg F$	`~F`
$F_1 \wedge \ldots \wedge F_n$	`F1 & ... & Fn`
$F_1 \vee \ldots \vee F_n$	`F1 | ... | Fn`
$F_1 \rightarrow F_n$	`F1 => Fn`
$(\forall x_1) \ldots (\forall x_n) F$	`! [X1,...,Xn] : F`
$(\exists x_1) \ldots (\exists x_n) F$	`? [X1,...,Xn] : F`

Example in the TPTP Syntax

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
      mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

Example in the TPTP Syntax

- Comments;

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
      mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

Example in the TPTP Syntax

- Comments;
- Input formula names;

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
      mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

Example in the TPTP Syntax

- Comments;
- Input formula names;
- Input formula roles (very important);

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
      mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

Example in the TPTP Syntax

- Comments;
- Input formula names;
- Input formula roles (very important);
- Equality

```
%---- 1 * x = x
fof(left_identity,axiom,(
  ! [X] : mult(e,X) = X )).
%---- i(x) * x = 1
fof(left_inverse,axiom,(
  ! [X] : mult(inverse(X),X) = e )).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,(
  ! [X,Y,Z] :
      mult(mult(X,Y),Z) = mult(X,mult(Y,Z)) )).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  ! [X] : mult(X,X) = e ).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  ! [X,Y] : mult(X,Y) = mult(Y,X) ).
```

Running a Theorem Prover on a TPTP file

is easy: for example

```
vampire <filename>
```

Running a Theorem Prover on a TPTP file

is easy: for example

```
vampire <filename>
```

One can also run Vampire with various options. For example, save the group theory problem in a file `group.tptp` and try

```
vampire                    group.tptp
```

Running a Theorem Prover on a TPTP file

is easy: for example

```
vampire <filename>
```

One can also run Vampire with various options. For example, save the group theory problem in a file `group.tptp` and try

```
vampire --thanks ECSS group.tptp
```

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                    [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                        [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                        [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                    [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                    [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                        [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                    [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus
- Proof by refutation, generating and simplifying inferences, unused formulas . . .

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                    [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus
- Proof by refutation, generating and simplifying inferences, unused formulas . . .

Proof by Vampire (Slightliy Modified)

```
Refutation found.
270. $false [trivial inequality removal 269]
269. mult(sk0,sk1) != mult (sk0,sk1) [superposition 14,125]
125. mult(X2,X3) = mult(X3,X2) [superposition 21,90]
90. mult(X4,mult(X3,X4)) = X3 [forward demodulation 75,27]
75. mult(inverse(X3),e) = mult(X4,mult(X3,X4)) [superposition 22,19]
27. mult(inverse(X2),e) = X2 [superposition 21,11]
22. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 17,10]
21. mult(X0,mult(X0,X1)) = X1 [forward demodulation 15,10]
19. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 12,13]
17. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 12,11]
15. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 12,13]
14. mult(sK0,sK1) != mult(sK1,sK0) [cnf transformation 9]
13. e = mult(X0,X0) [cnf transformation 4]
12. mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [cnf transformation 3]
11. e = mult(inverse(X0),X0) [cnf transformation 2]
10. mult(e,X0) = X0 [cnf transformation 1]
9. mult(sK0,sK1) != mult(sK1,sK0) [skolemisation 7,8]
8. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) <=> mult(sK0,sK1) != mult(sK1,sK0)
                                                    [choice axiom]
7. ?[X0,X1]: mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~![X0,X1]: mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ![X0,X1]: mult(X0,X1) = mult(X1,X0) [input]
4. ![X0]: e = mult(X0,X0)[input]
3. ![X0,X1,X2]: mult(X0,mult(X1,X2)) = mult(mult(X0,X1),X2) [input]
2. ![X0]: e = mult(inverse(X0),X0) [input]
1. ![X0]: mult(e,X0) = X0 [input]
```

- Each inference derives a formula from zero or more other formulas;
- Input, preprocessing, new symbols introduction, superposition calculus
- Proof by refutation, generating and simplifying inferences, unused formulas ...

Vampire

- Completely automatic: once you started a proof attempt, it can only be interrupted by terminating the process.

Vampire

- Completely automatic: once you started a proof attempt, it can only be interrupted by terminating the process.
- Champion of the CASC world-cup in first-order theorem proving: won CASC 38 times.

What an Automated Theorem Prover is Expected to Do

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).

Proof by Refutation

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

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

Proof by Refutation

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

1. negate the conjecture ($\neg G$);
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.

Proof by Refutation

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

1. negate the conjecture ($\neg G$);
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.

In this formulation the negation of the conjecture $\neg G$ is treated like any other formula.

In fact, Vampire (and other provers) internally treat conjectures differently, to make proof search more goal-oriented.

General Scheme (simplified)

- Read a problem;
- Determine proof-search options to be used for this problem;
- Preprocess the problem;
- Convert it into a normal form (CNF);
- Run a saturation algorithm on it, try to derive *false*.
- If *false* is derived, report the result, maybe including a refutation.

General Scheme (simplified)

- Read a problem;
- Determine proof-search options to be used for this problem;
- Preprocess the problem;
- Convert it into a normal form (CNF);
- Run a saturation algorithm on it, try to derive *false*.
- If *false* is derived, report the result, maybe including a refutation.

Trying to derive *false* using a saturation algorithm is the hardest part, which in practice may not terminate or run out of memory.

How to Establish Unsatisfiability?

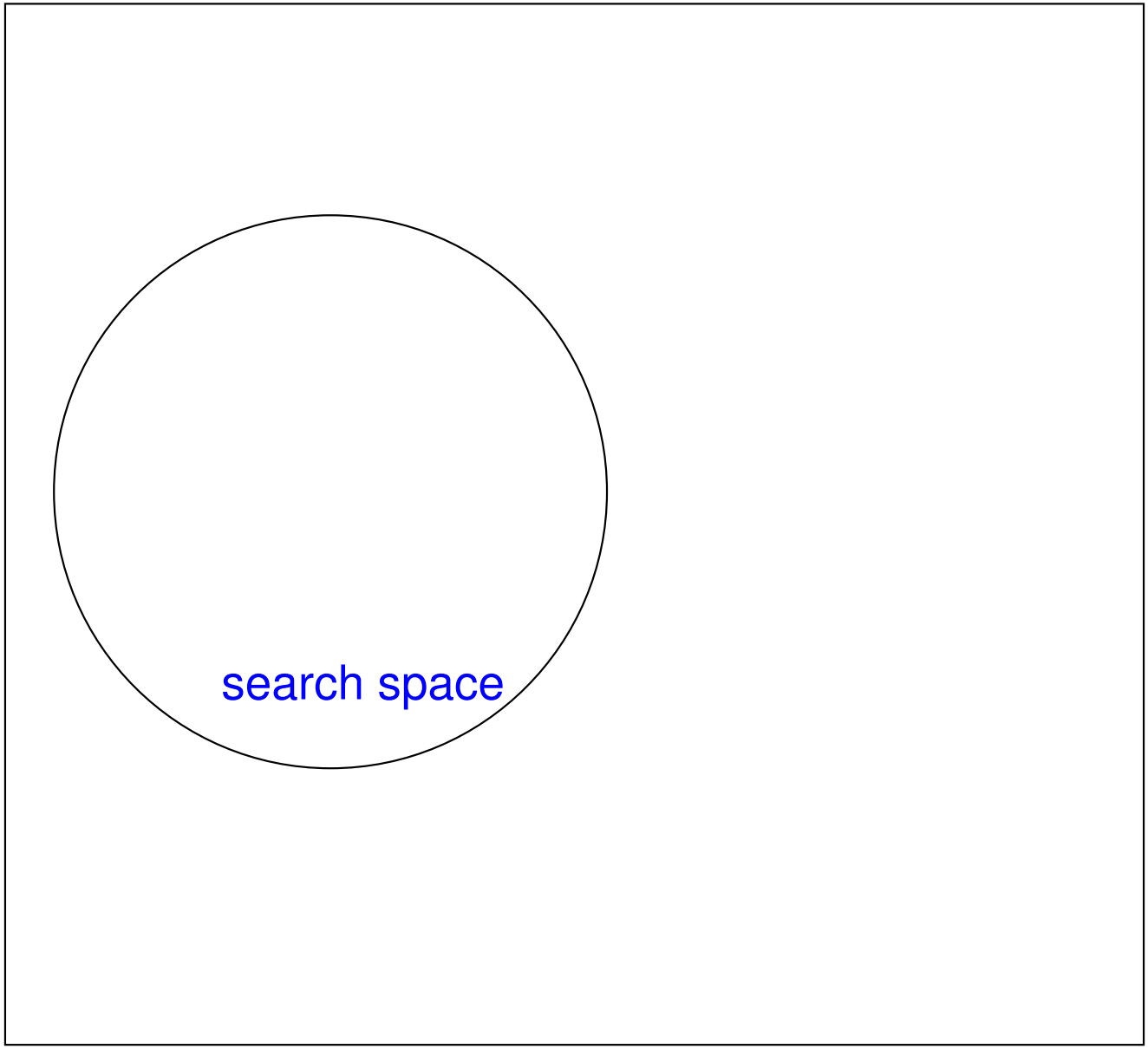
Idea:

- Take a set of clauses S (the search space), initially $S = S_0$.

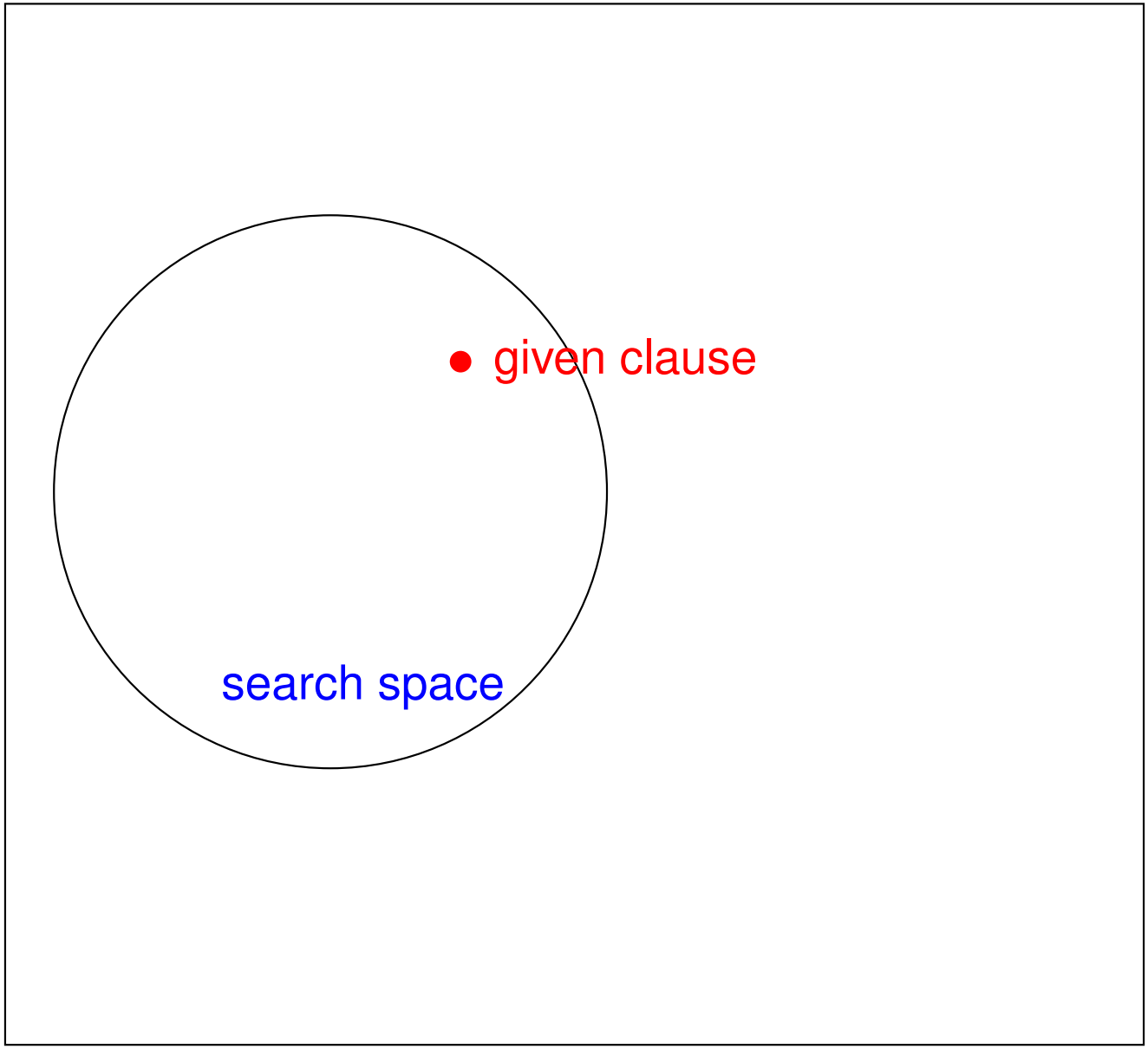
  Repeatedly apply inferences to clauses in S and add their conclusions to S, unless these conclusions are already in S.

- If, at any stage, we obtain *false*, we terminate and report unsatisfiability of S_0.

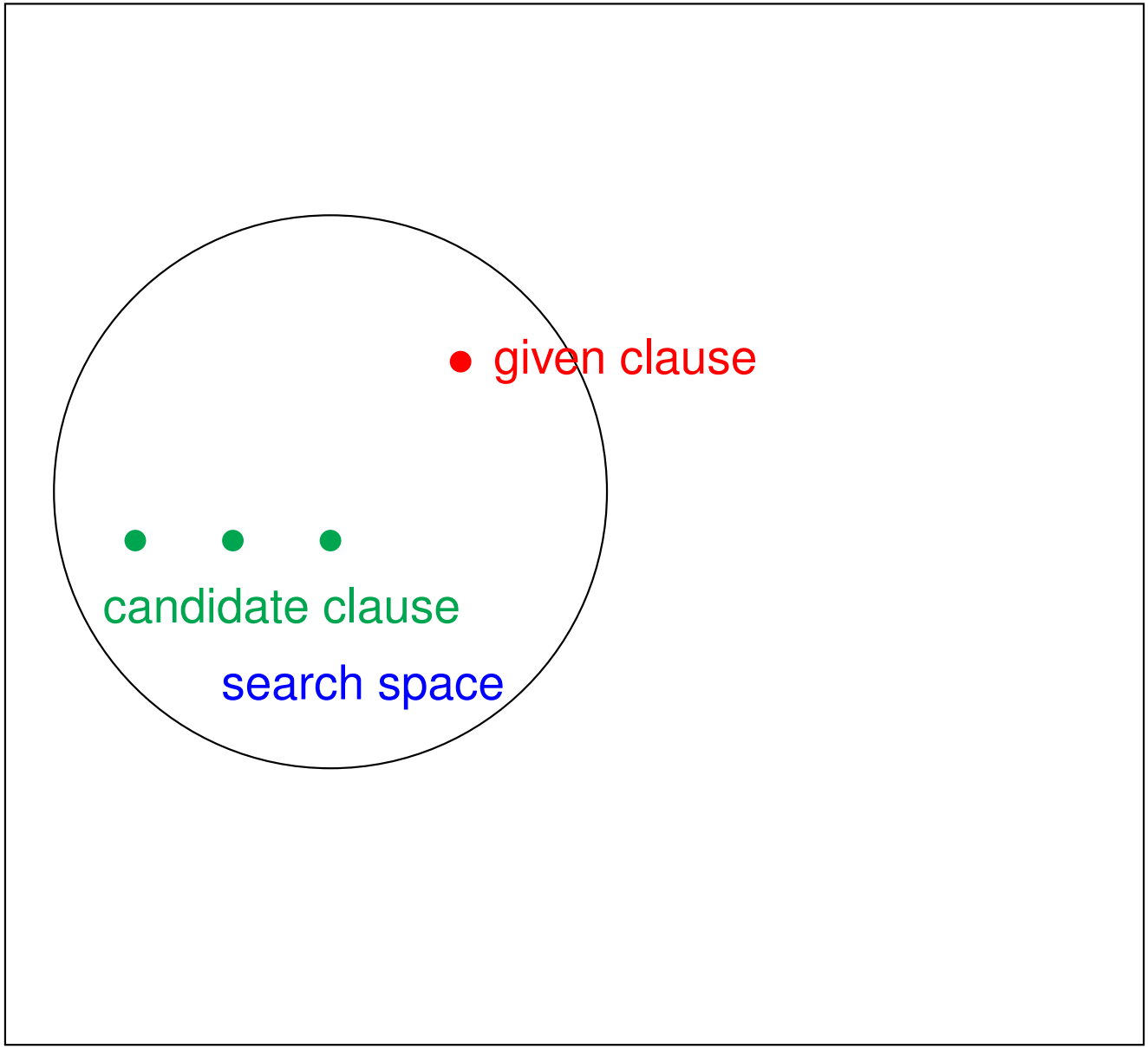
Saturation Algorithms



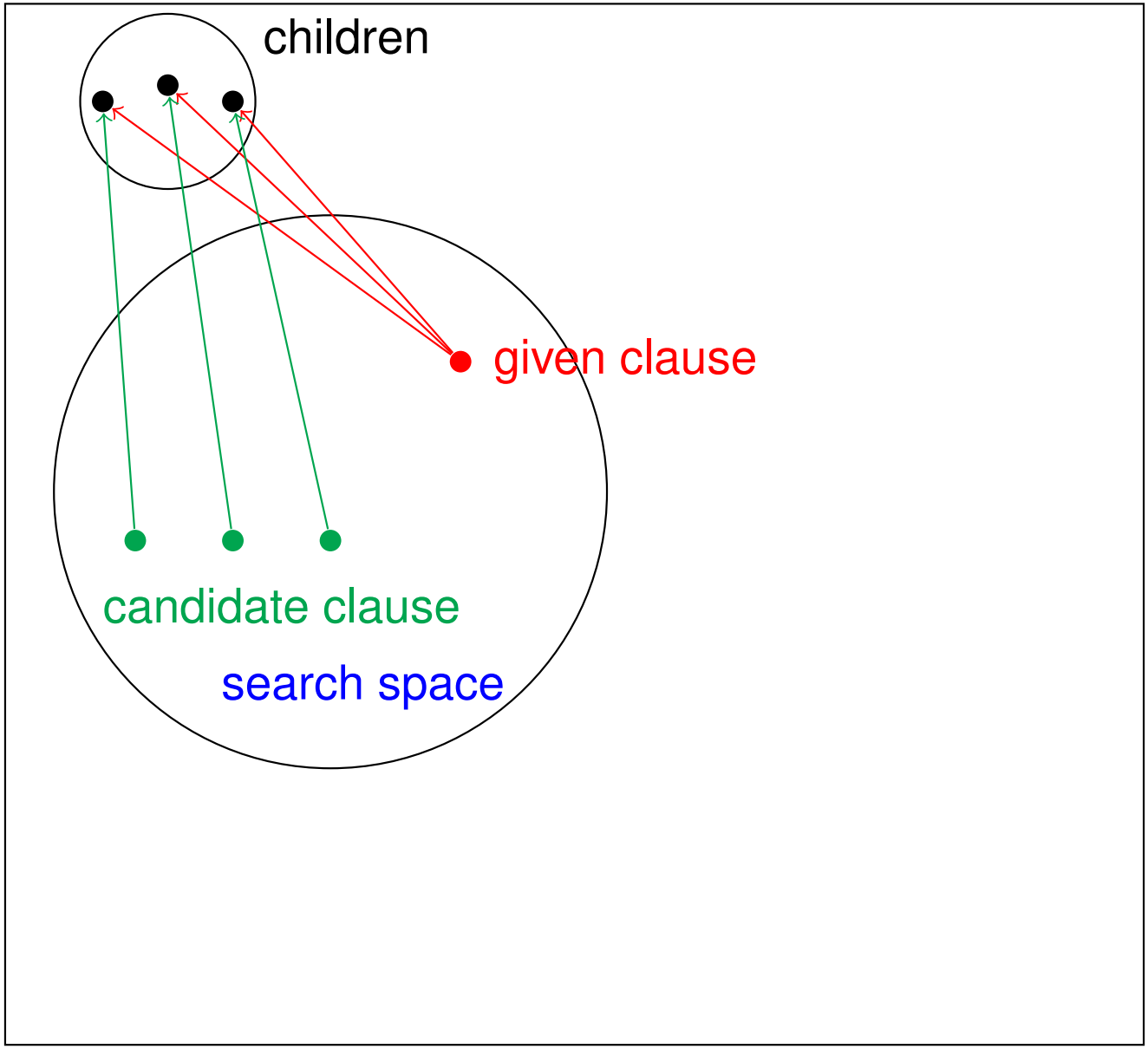
Saturation Algorithms



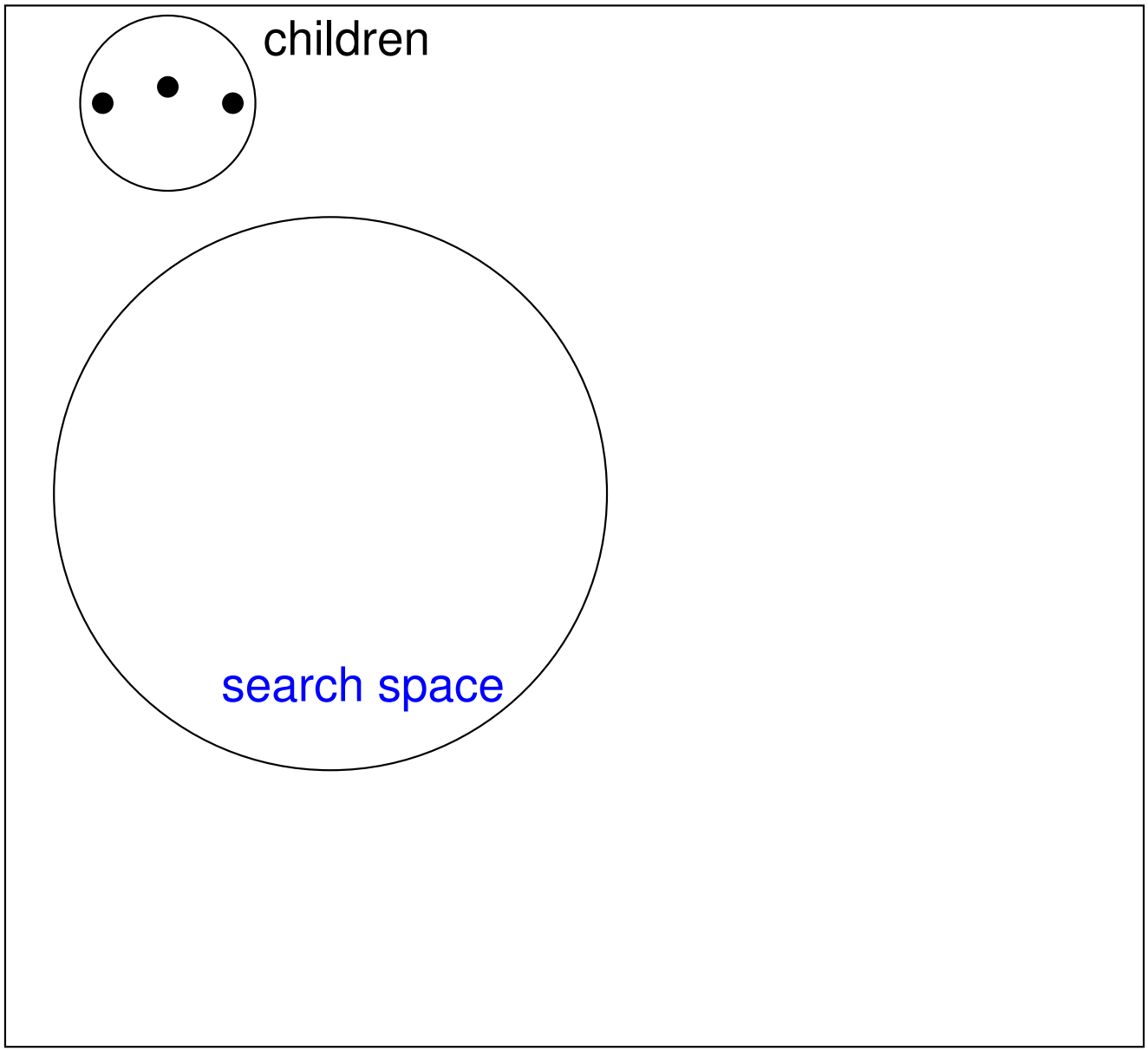
Saturation Algorithms



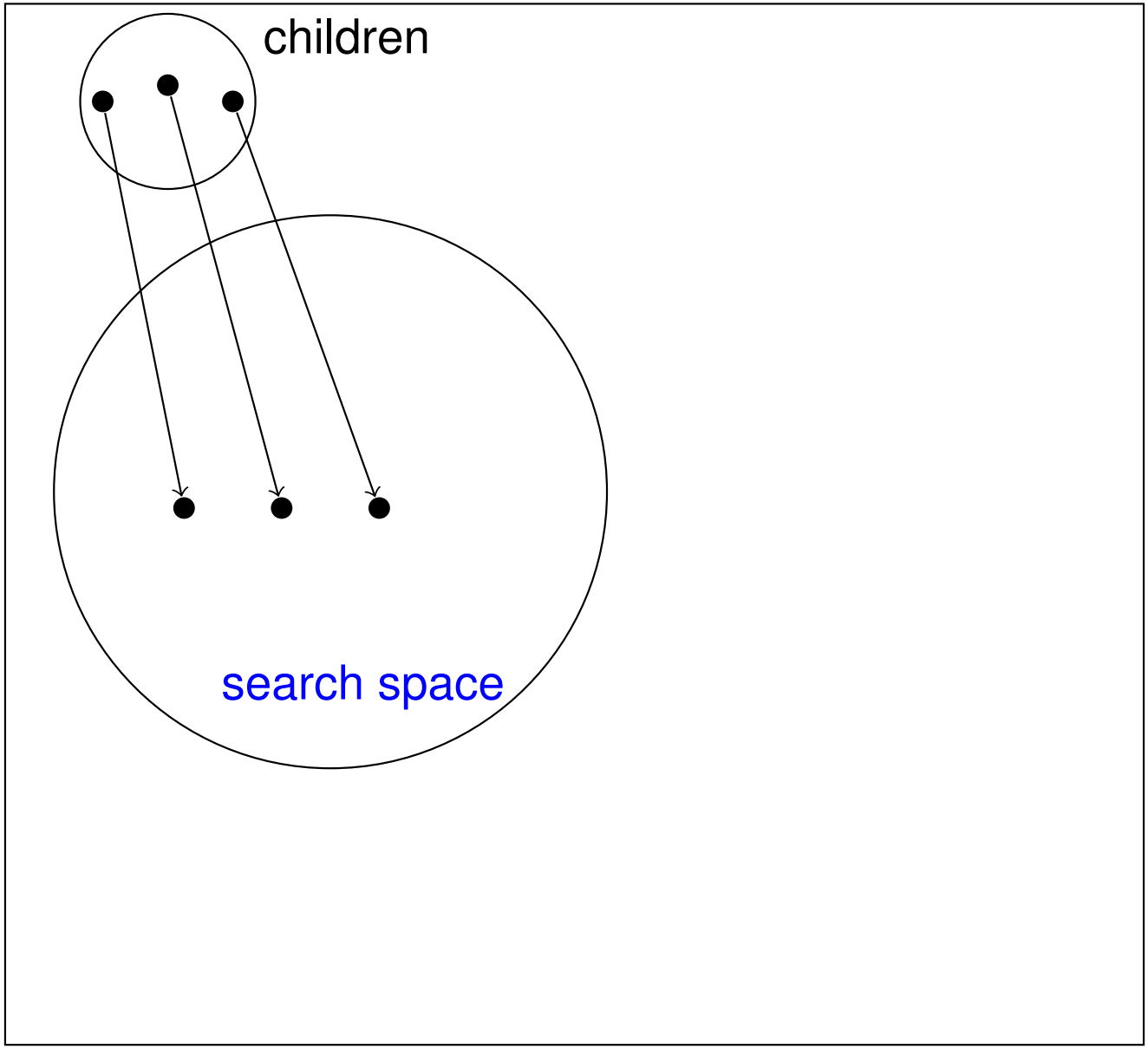
Saturation Algorithms



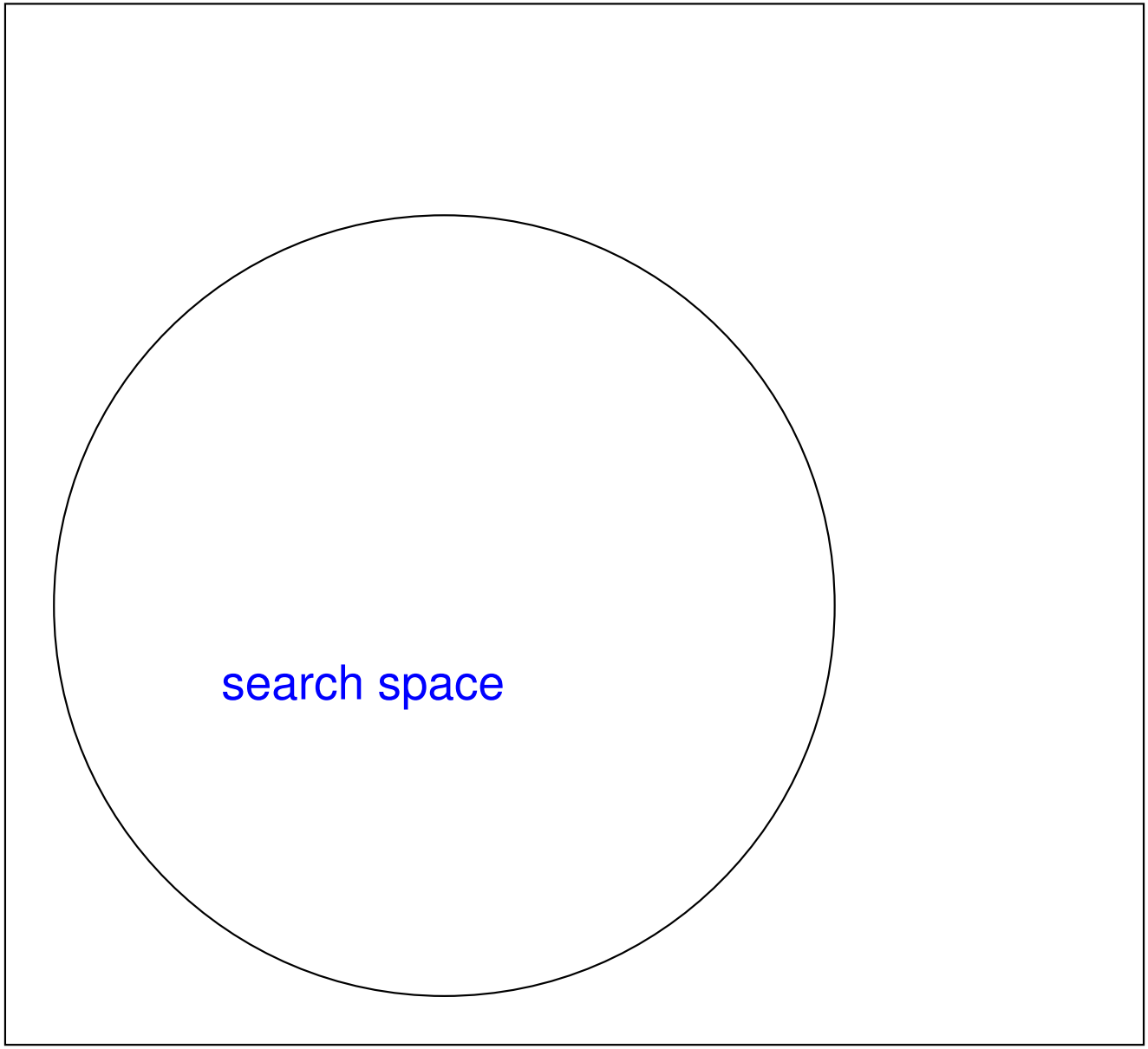
Saturation Algorithms



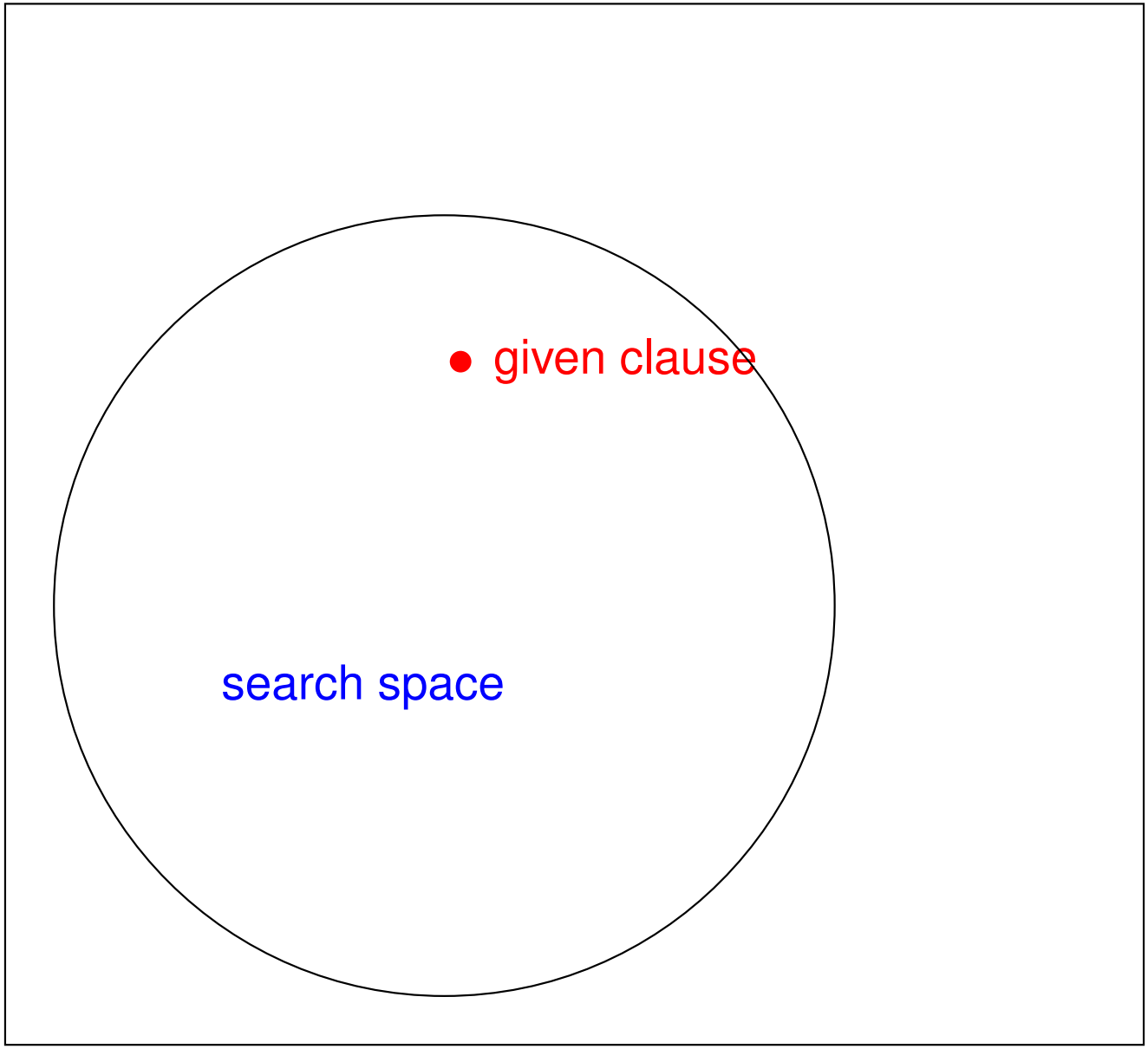
Saturation Algorithms



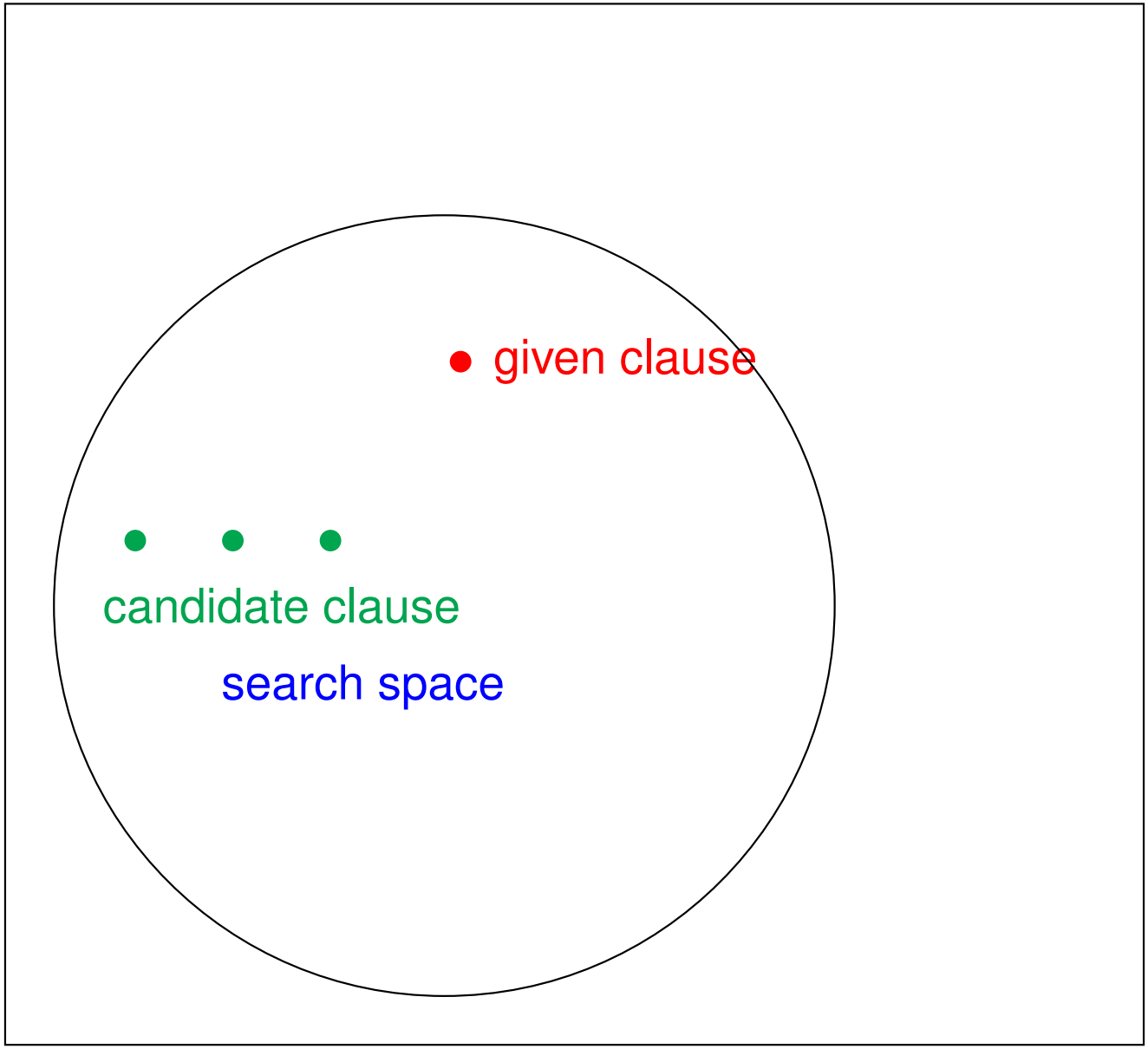
Saturation Algorithms



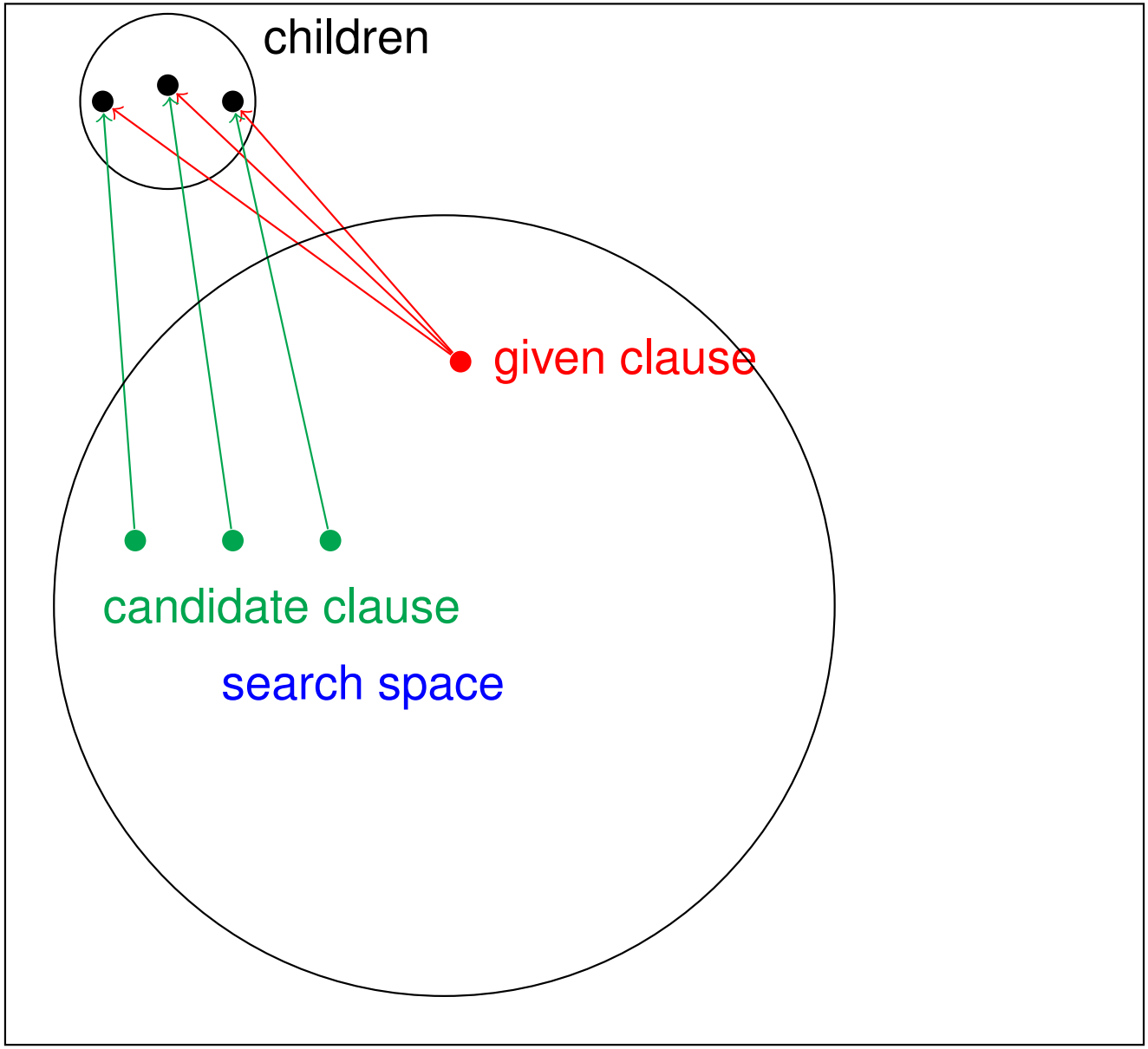
Saturation Algorithms



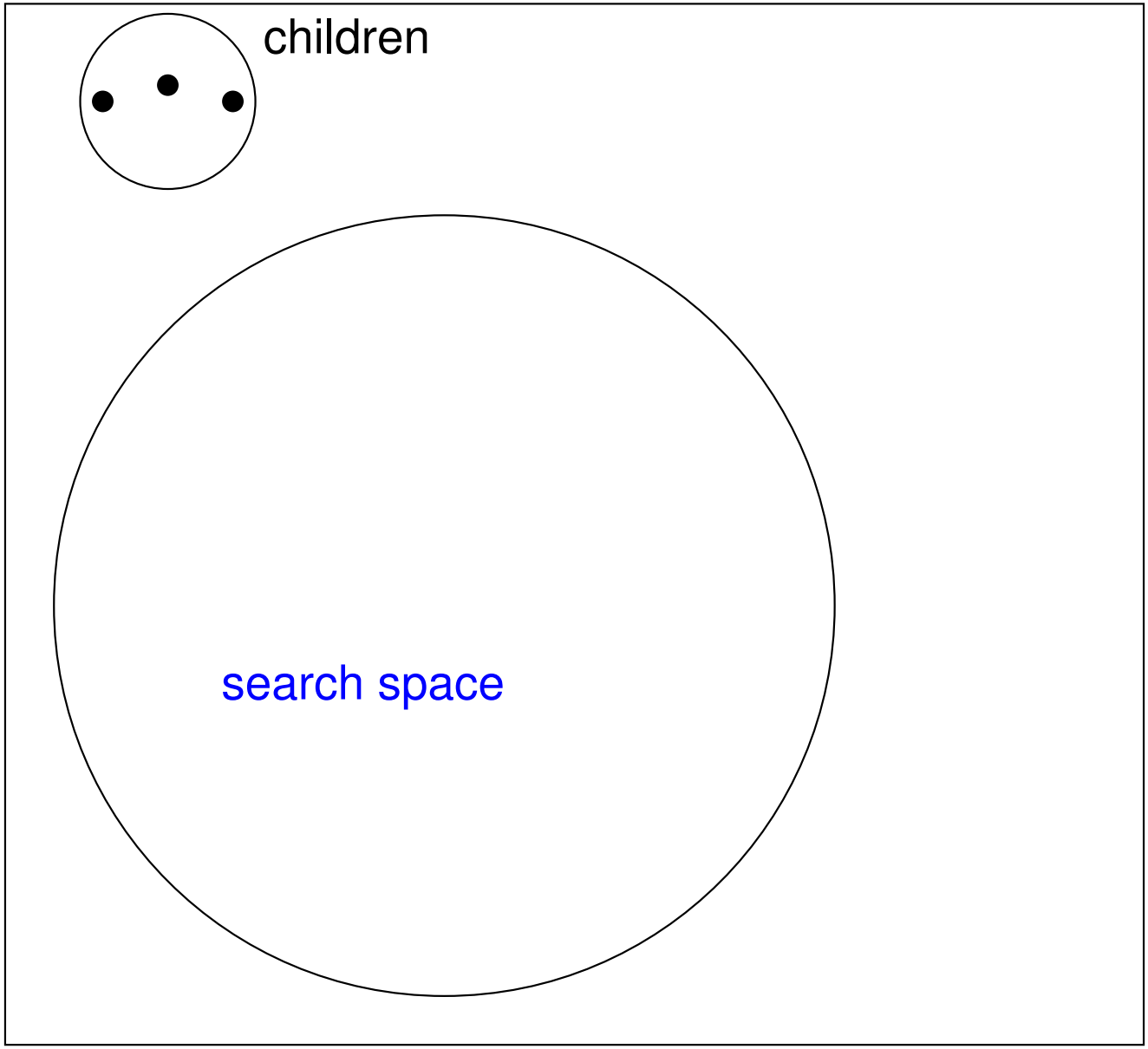
Saturation Algorithms



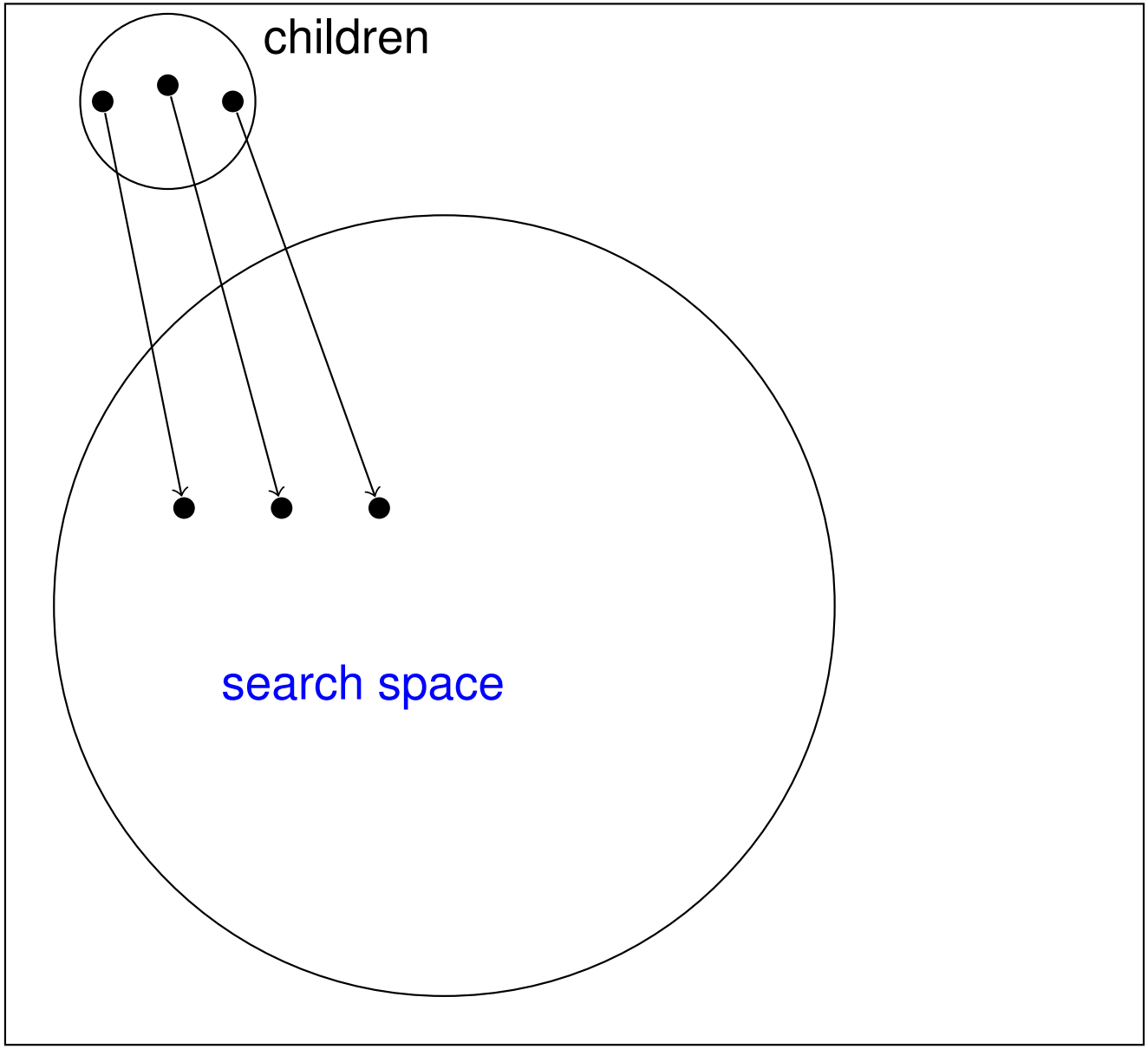
Saturation Algorithms



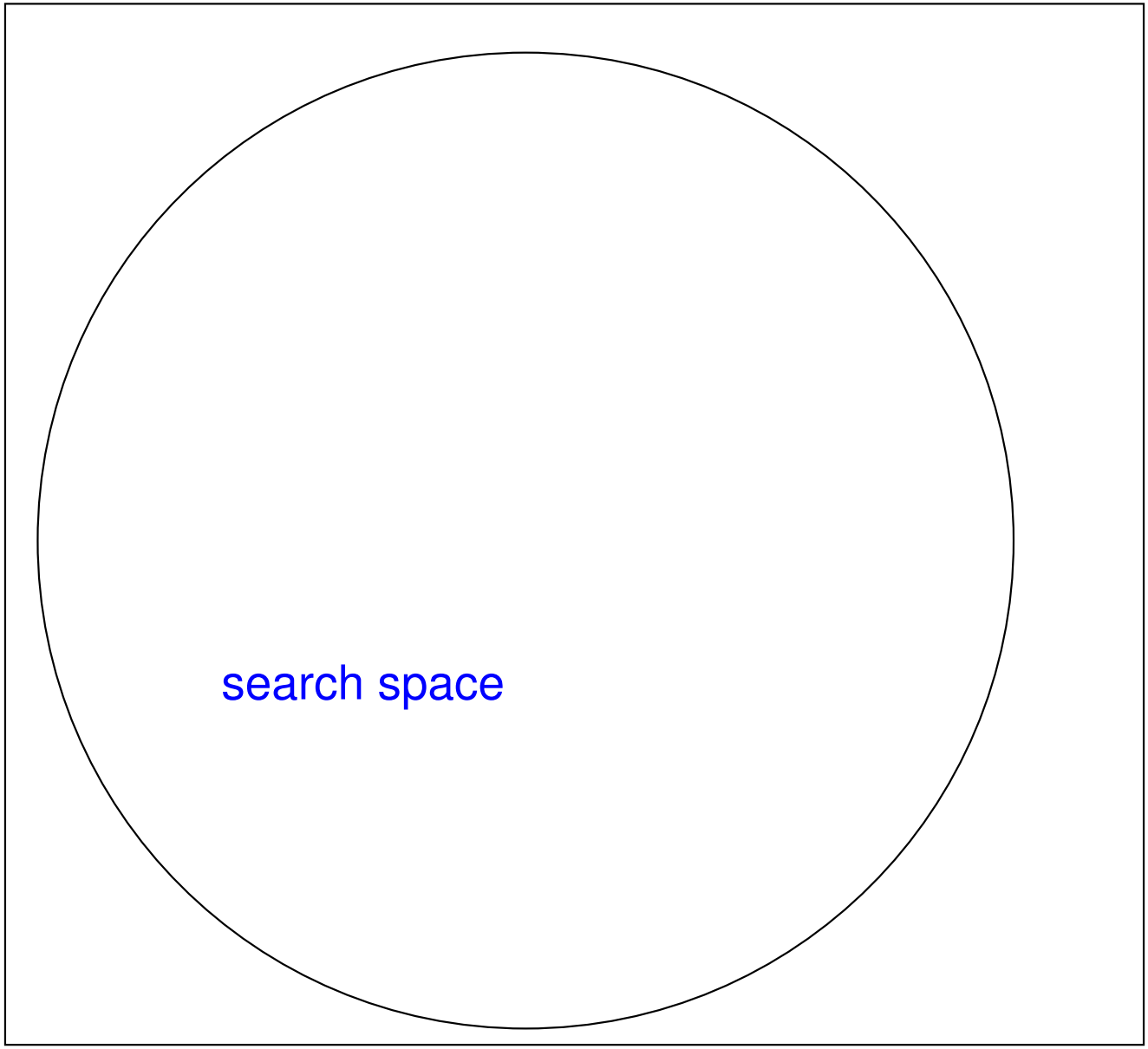
Saturation Algorithms



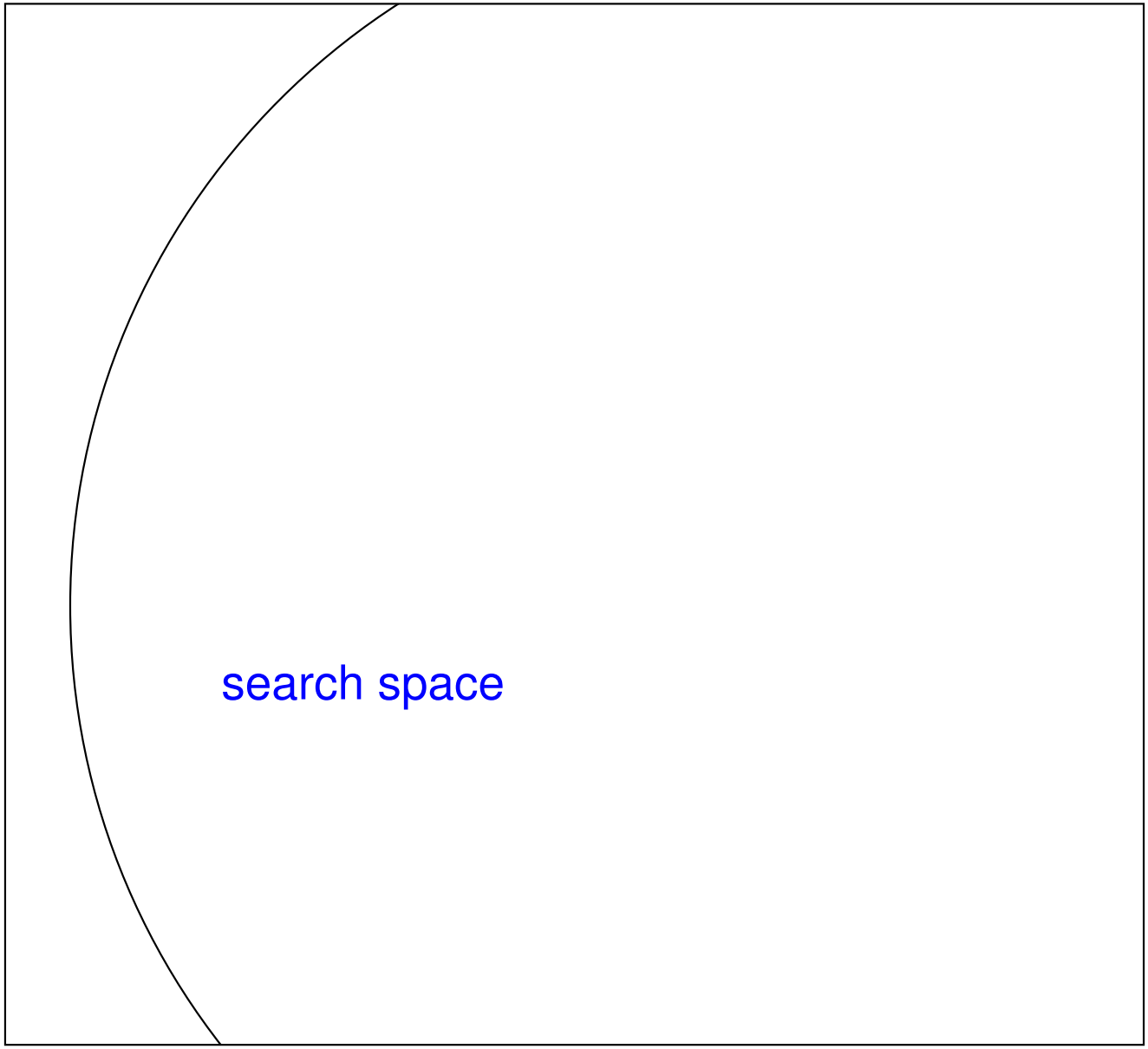
Saturation Algorithms



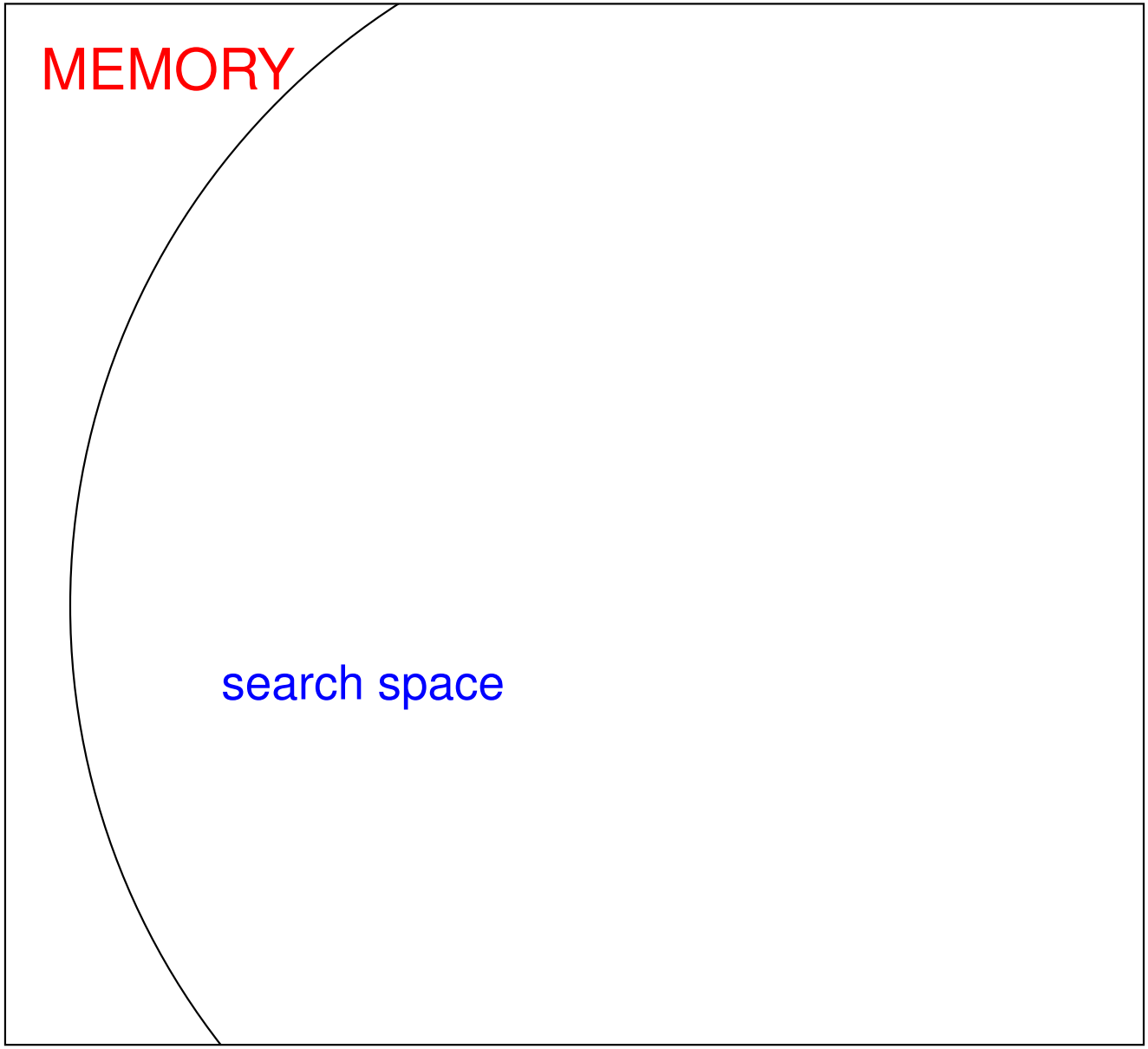
Saturation Algorithms



Saturation Algorithms



Saturation Algorithms



Saturation Algorithm in Practice

In practice there are three possible scenarios:

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

From Theory to Practice

In practice, saturation theorem provers implement:

- Preprocessing and CNF transformation;
- Superposition system;
- Orderings and selection functions;
- Fairness (saturation algorithms);
- Deletion and generation of clauses in the search space;
- Many, many proof options and stragegies

From Theory to Practice

In practice, saturation theorem provers implement:

- Preprocessing and CNF transformation;
- Superposition system;
- Orderings and selection functions;
- Fairness (saturation algorithms);
- Deletion and generation of clauses in the search space;
- Many, many proof options and stragegies

.

From Theory to Practice

In practice, saturation theorem provers implement:

- Preprocessing and CNF transformation;
- Superposition system;
- Orderings and selection functions;
- Fairness (saturation algorithms);
- Deletion and generation of clauses in the search space;
- Many, many proof options and stragegies
  – example: limited resource strategy.

From Theory to Practice

In practice, saturation theorem provers implement:

- Preprocessing and CNF transformation;
- Superposition system;
- Orderings and selection functions;
- Fairness (saturation algorithms);
- Deletion and generation of clauses in the search space;
- Many, many proof options and stragegies
  – example: limited resource strategy.

Try:

```
vampire --age_weight_ratio 10:1
  --time_limit 86400
      GRP140-1.p
```

Outline

Automated Theorem Prover was Expected to Do

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).

What an Automated Theorem Prover is Expected to Do

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

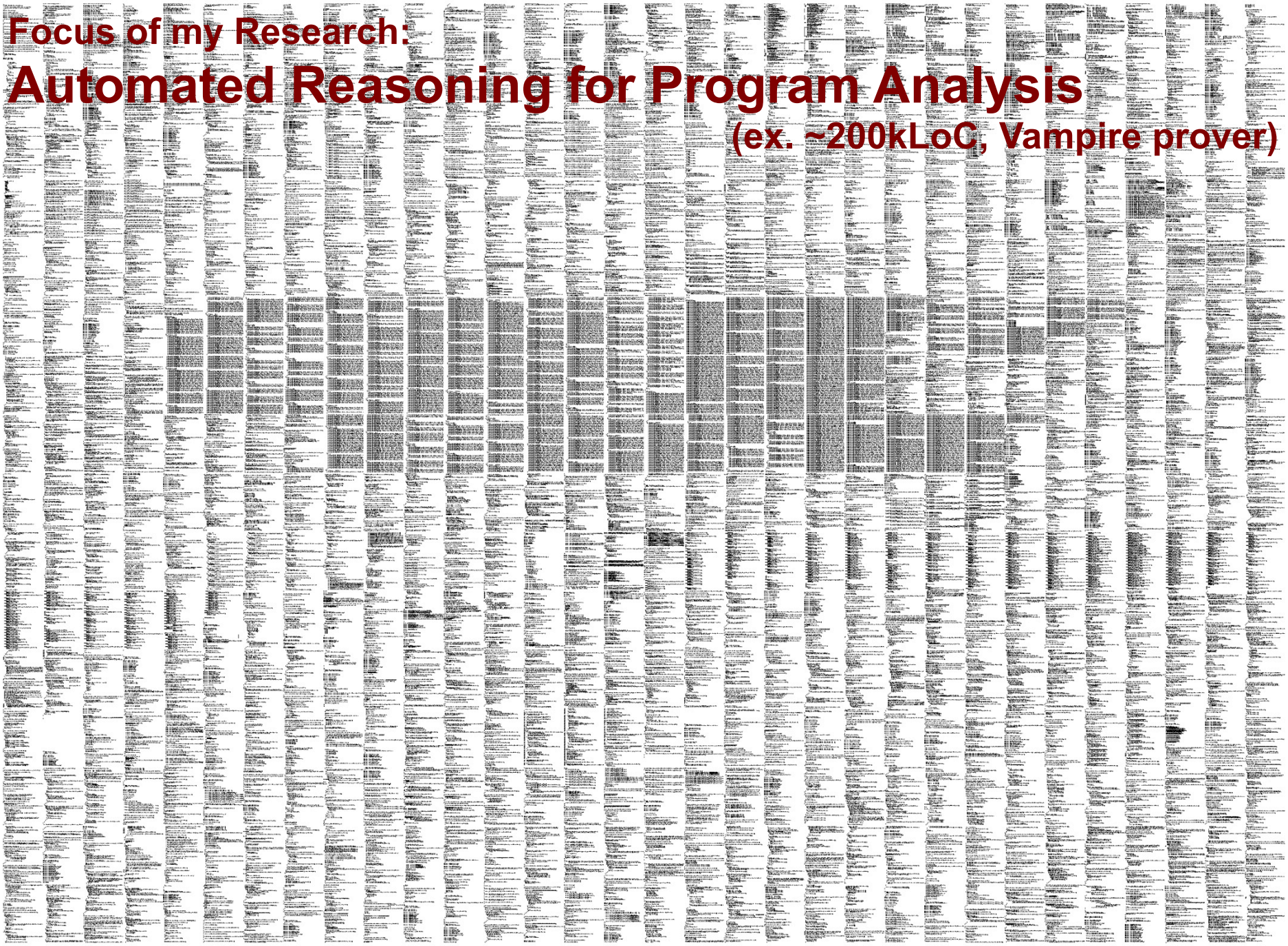
Output:

- readable proof;
- relevant lemmas extracted from proofs;
- Craig interpolants extraced from software safety proofs;
- program analysis;
- invariant generation;
- inductive reasoning;
- Reasoning with first-order theories of data structures;
- ...

Focus of my Research:

Automated Reasoning for Program Analysis

(ex. ~200kLoC, Vampire prover)



Focus of my Research:

Automated Reasoning for Program Analysis

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Focus of my Research:

Automated Reasoning for Program Analysis

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Program property:

$$(\forall p)(0 \le p < b \Rightarrow (\exists q)(0 \le q < a \wedge B[p]=A[q]+h(p) \wedge A[q]>0)$$

Focus of my Research:

Automated Reasoning for Program Analysis

```
cnt=0, fib1=1, fib2=0;
while (cnt<n) do

t=fib1; fib1=fib1+fib2; fib2=t; cnt++;

end do
```

h

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Focus of my Research:

Automated Reasoning for Program Analysis

```
cnt=0, fib1=1, fib2=0;
while (cnt<n) do

t=fib1; fib1=fib1+fib2; fib2=t; cnt++;

end do
```

h

Program property:

$fib1^4 + fib2^4 + 2*fib1*fib2^3 - 2\ fib1^3*fib2 - fib1^2*fib2^2 - 1 = 0$

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```

Focus of my Research:

Automated Reasoning for Program Analysis

```
cnt=0, fib1=1, fib2=0;
while (cnt<n) do

t=fib1; fib1=fib1+fib2; fib2=t; cnt++;

end do
```

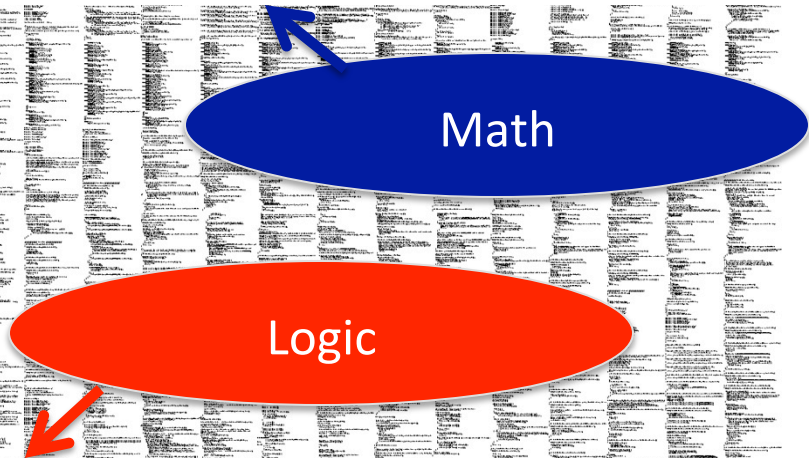
h

$fib1^4 + fib2^4 + 2*fib1*fib2^3 - 2\ fib1^3*fib2 - fib1^2*fib2^2 - 1 = 0$

```
a=0, b=0, c=0;
while (a<n) do

if A[a]>0 then B[b]=A[a]+h(b); b=b+1;
          else C[c]=A[a]; c=c+1;

a=a+1;
end do
```



$(\forall p)(0 \le p < b \Rightarrow$

$(\exists q)(0 \le q < a \wedge B[p]=A[q]+h(p) \wedge A[q]>0)$

Math

Logic

My Research

Program Analysis

Symbolic Computation

Automated Theorem Proving

My Research

funded by:

Program Analysis