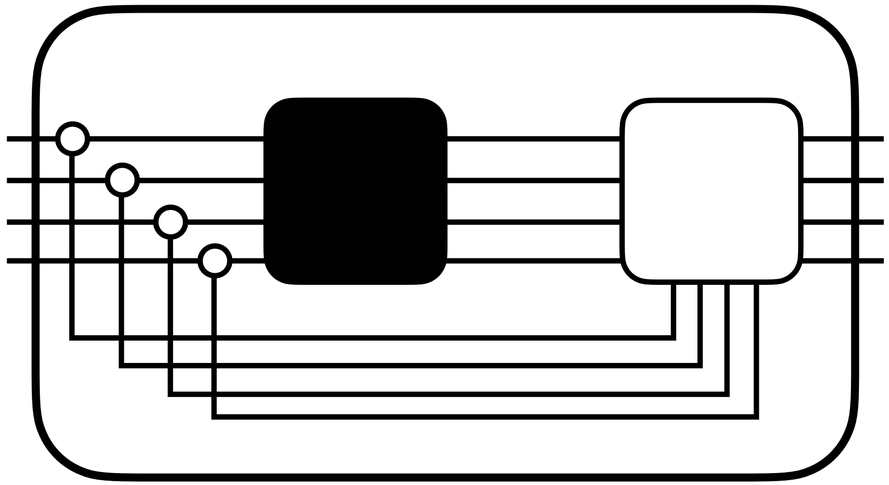
Quantified Boolean Formulas

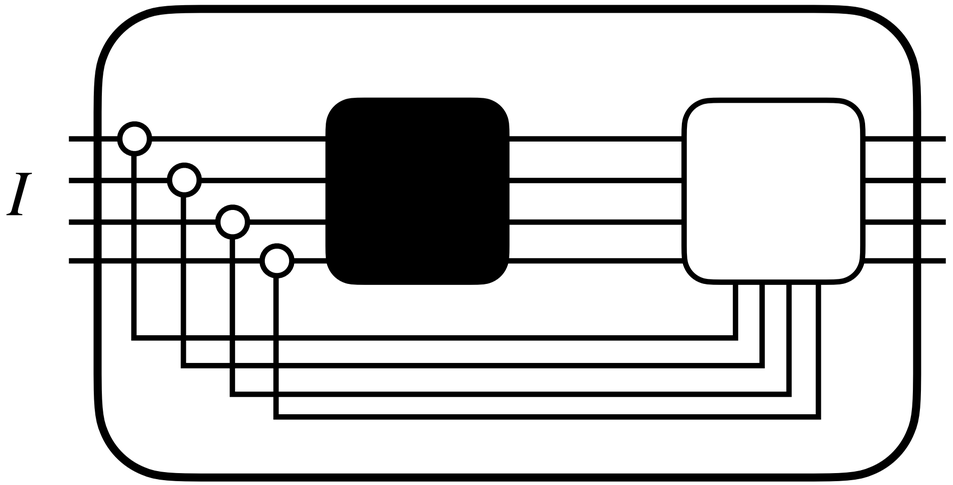
Friedrich Slivovsky

Indian SAT+SMT School 2024

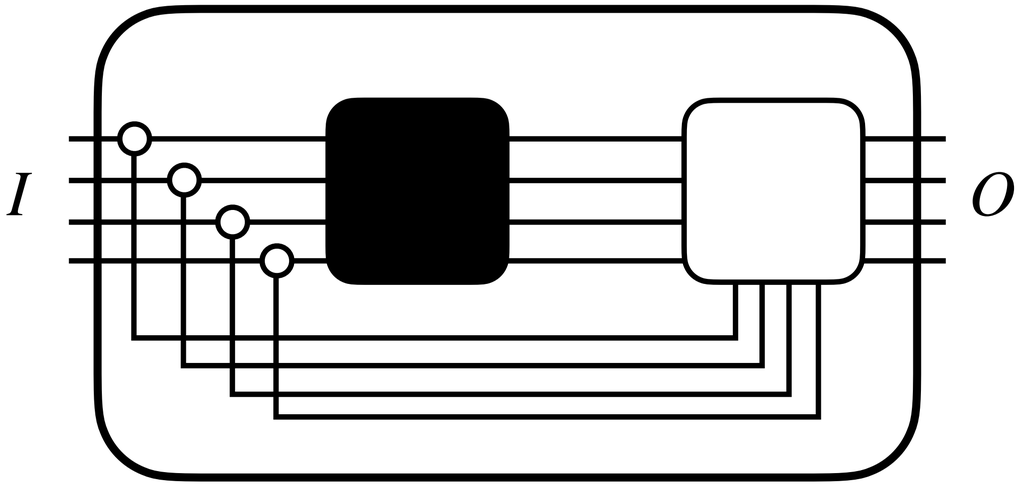
Limits of SAT



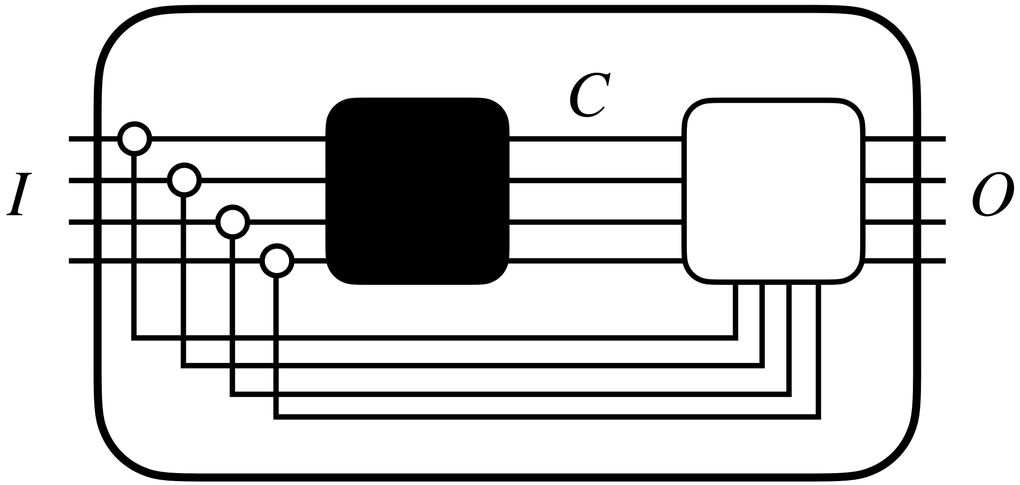
Limits of SAT



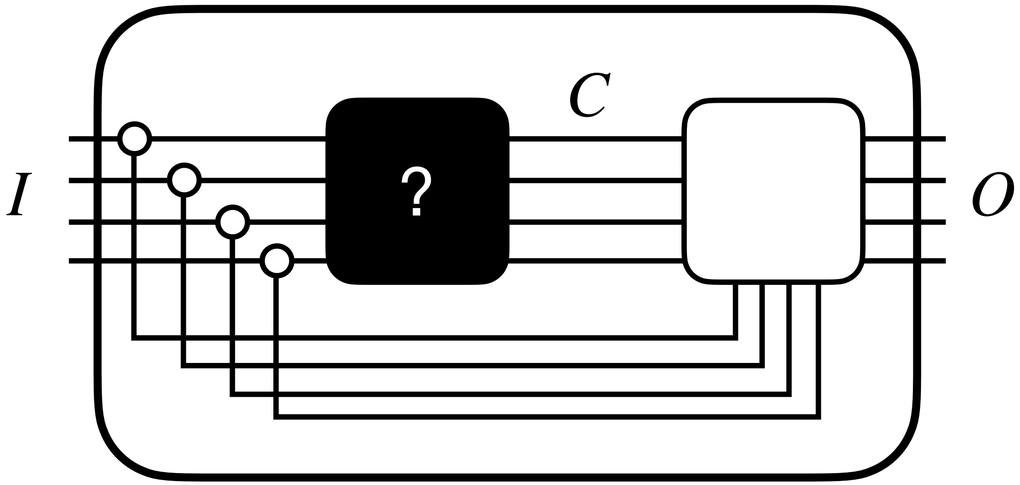
Limits of SAT



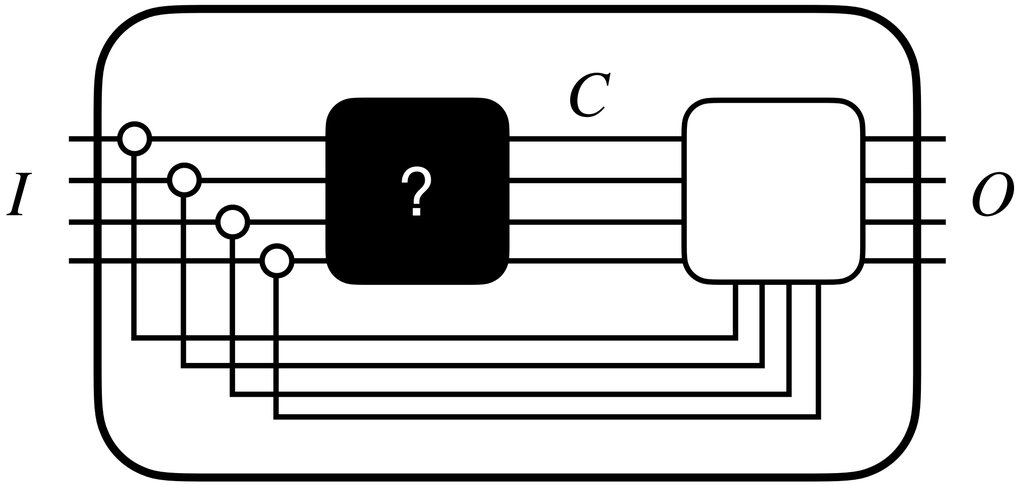
Limits of SAT



Limits of SAT

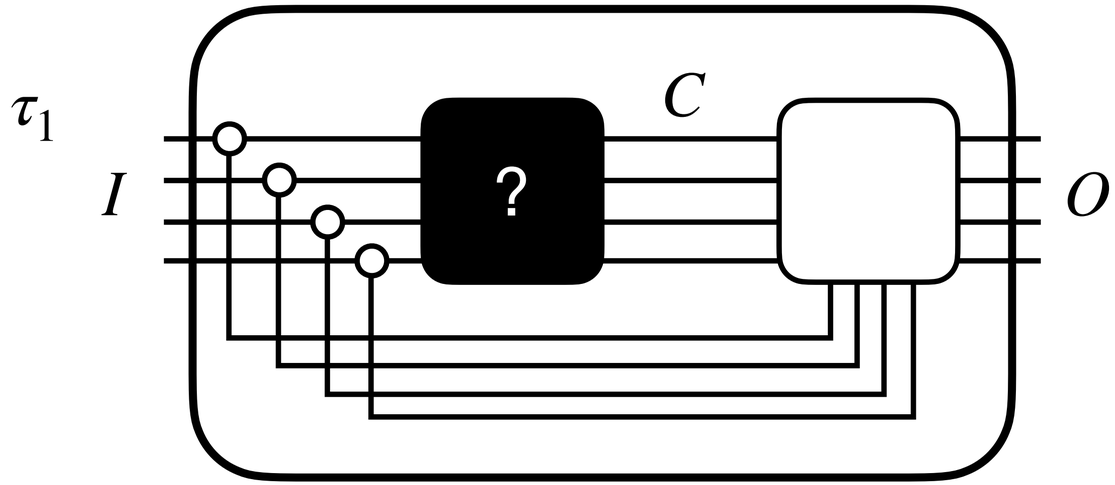


Limits of SAT



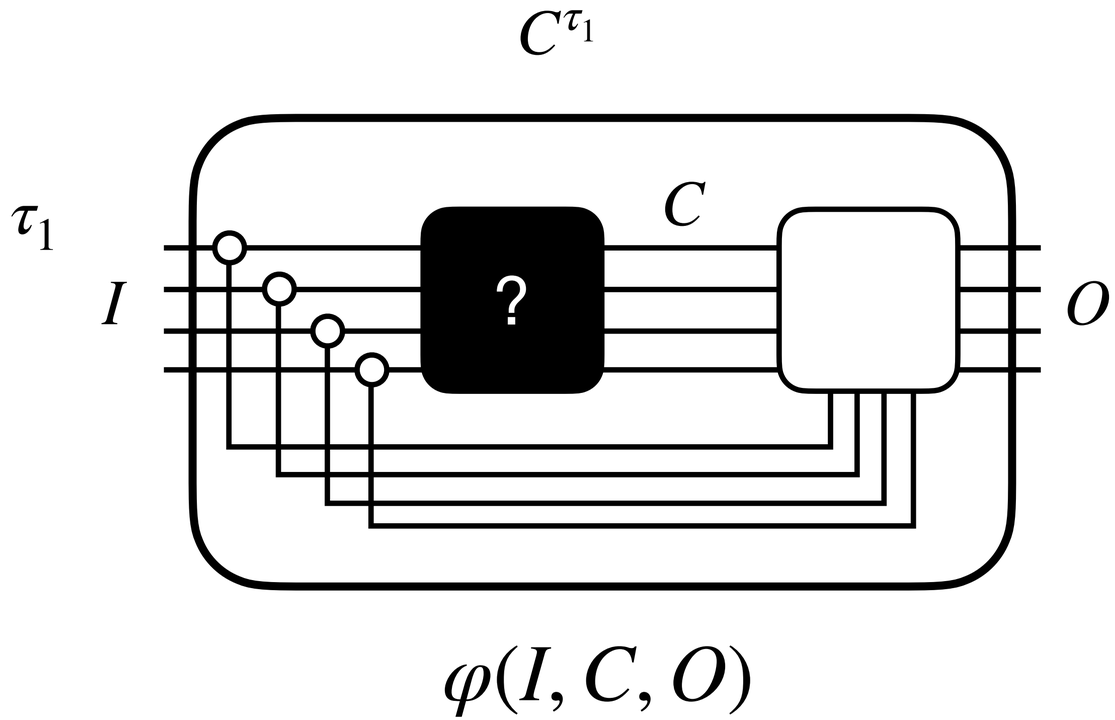
$$\varphi(I, C, O)$$

Limits of SAT

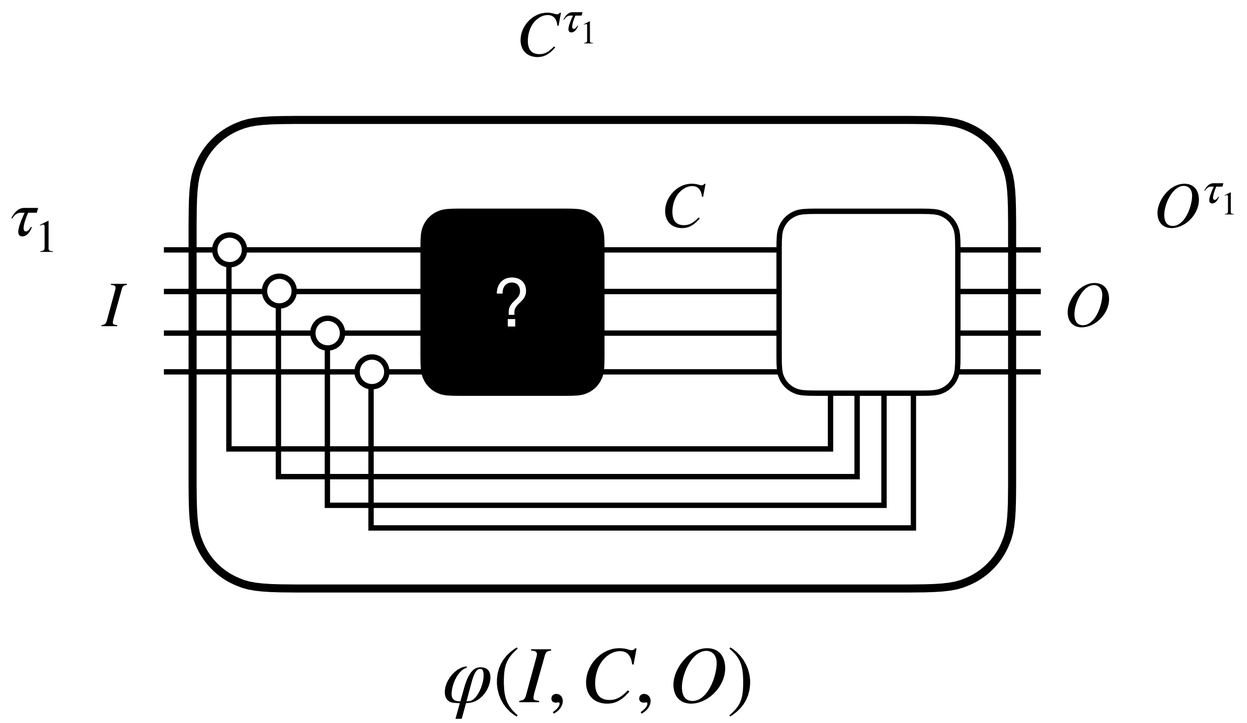


$\varphi(I, C, O)$

Limits of SAT

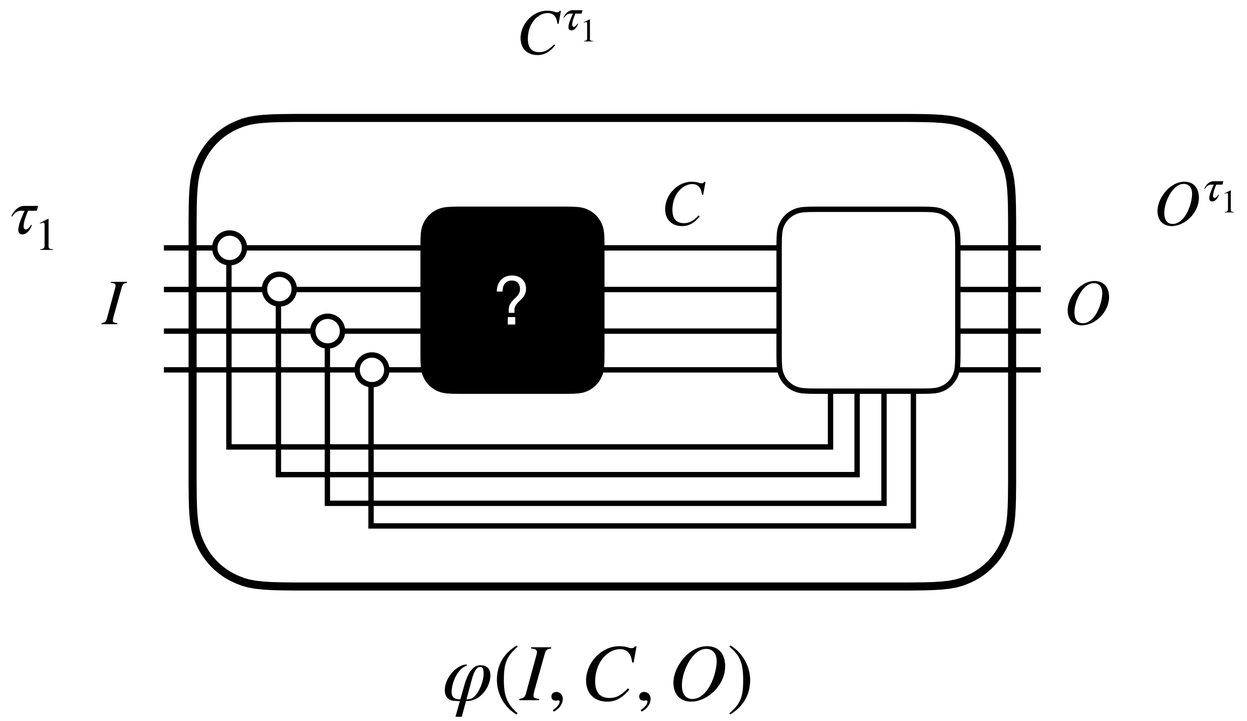


Limits of SAT

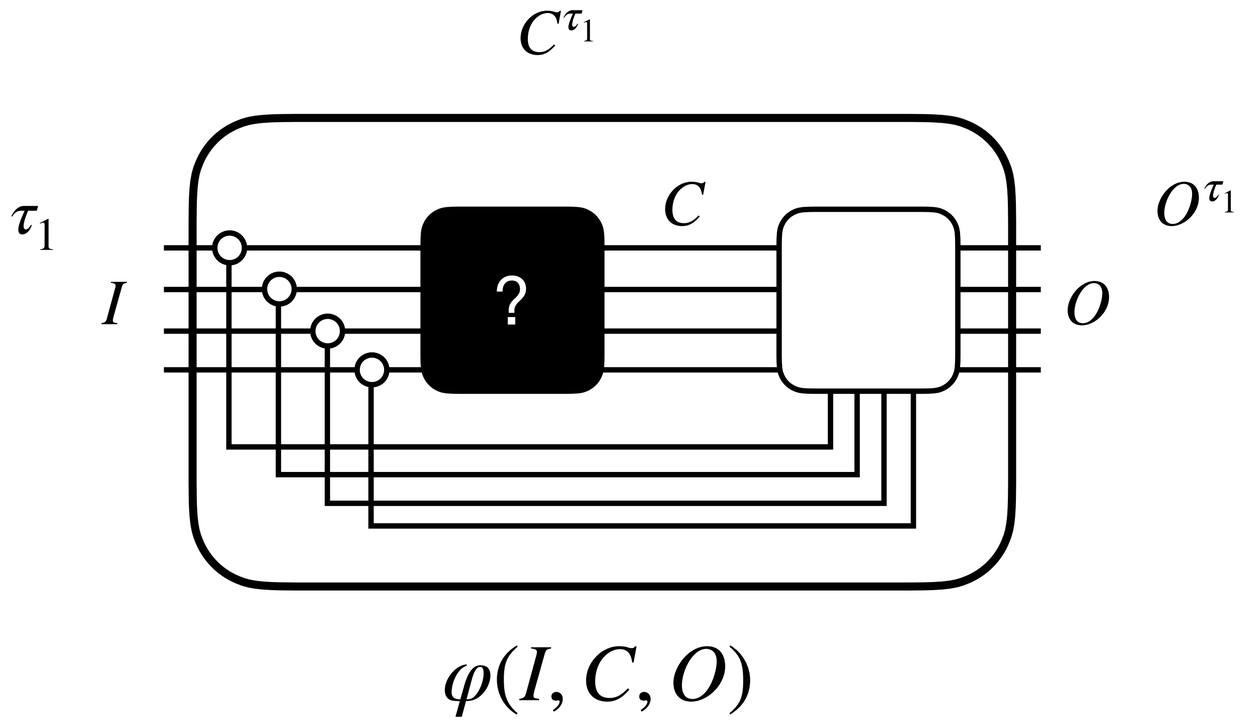


Limits of SAT

SAT encoding



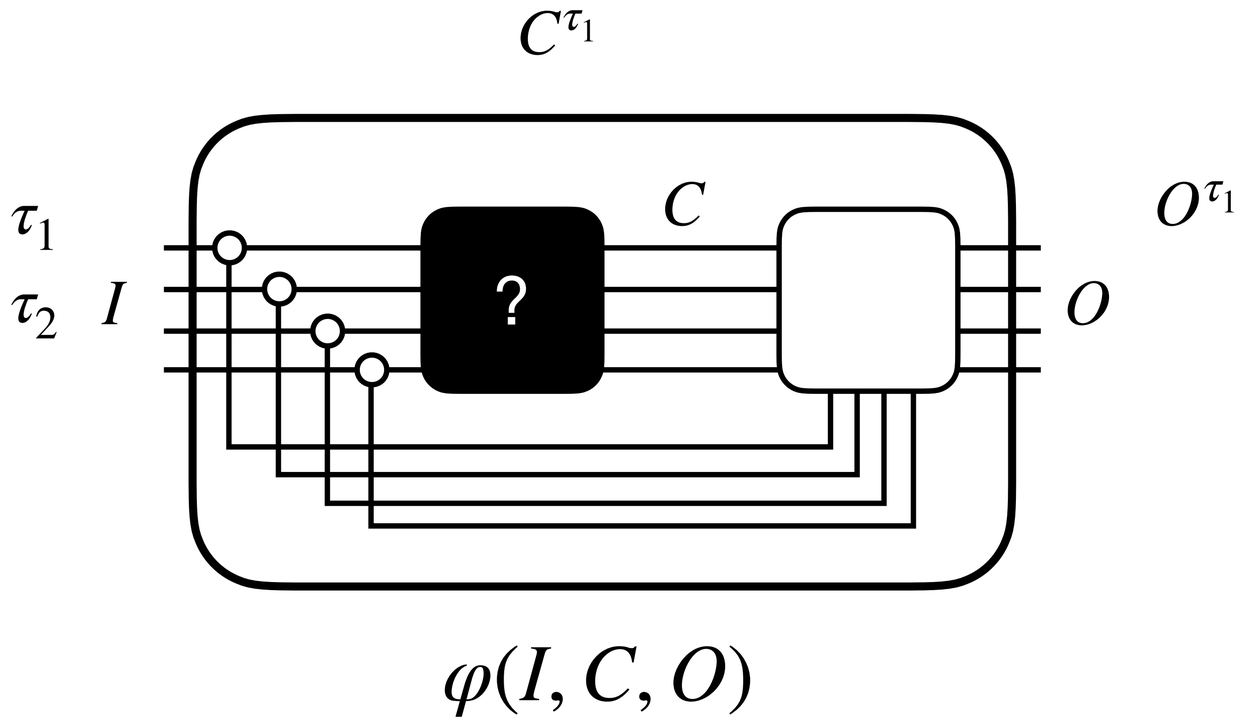
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$

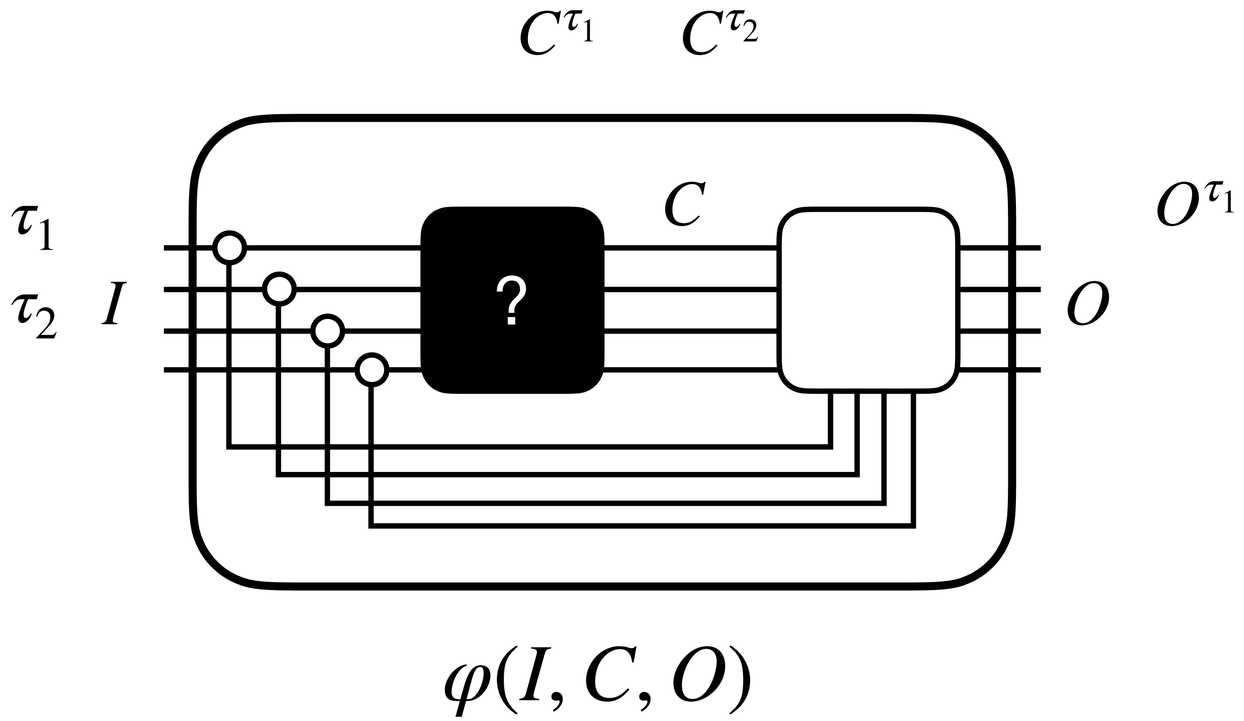
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$

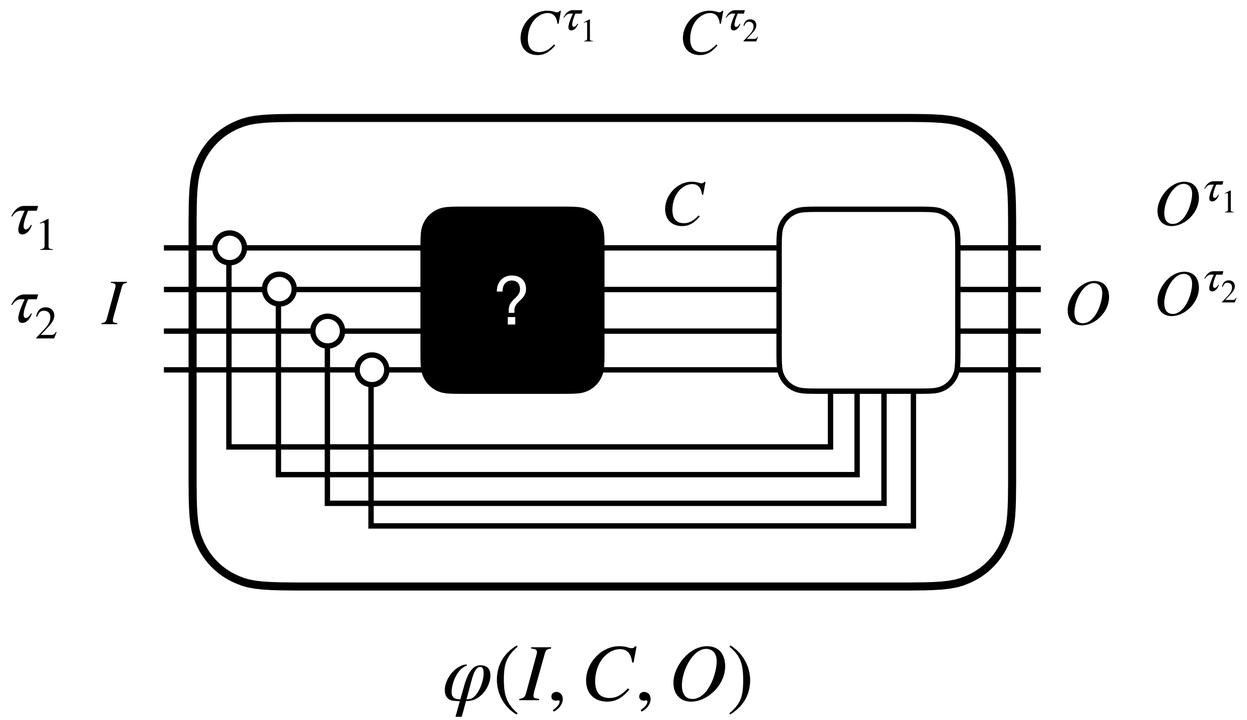
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$

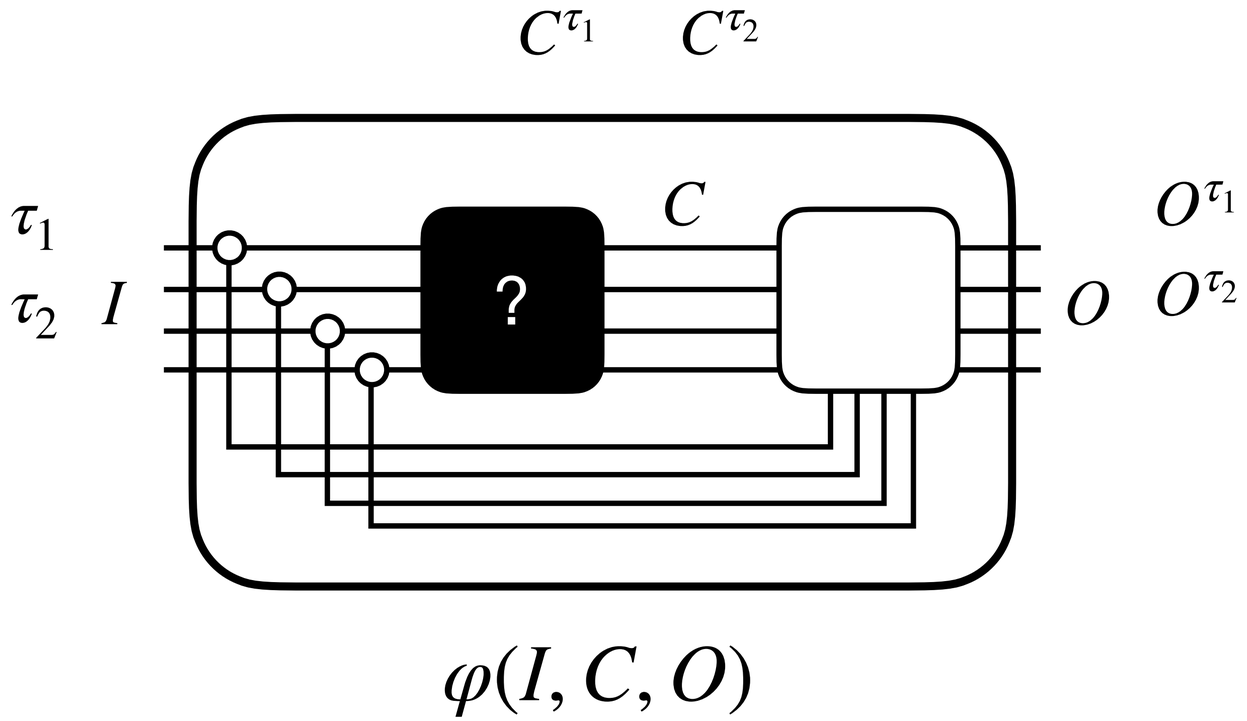
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$

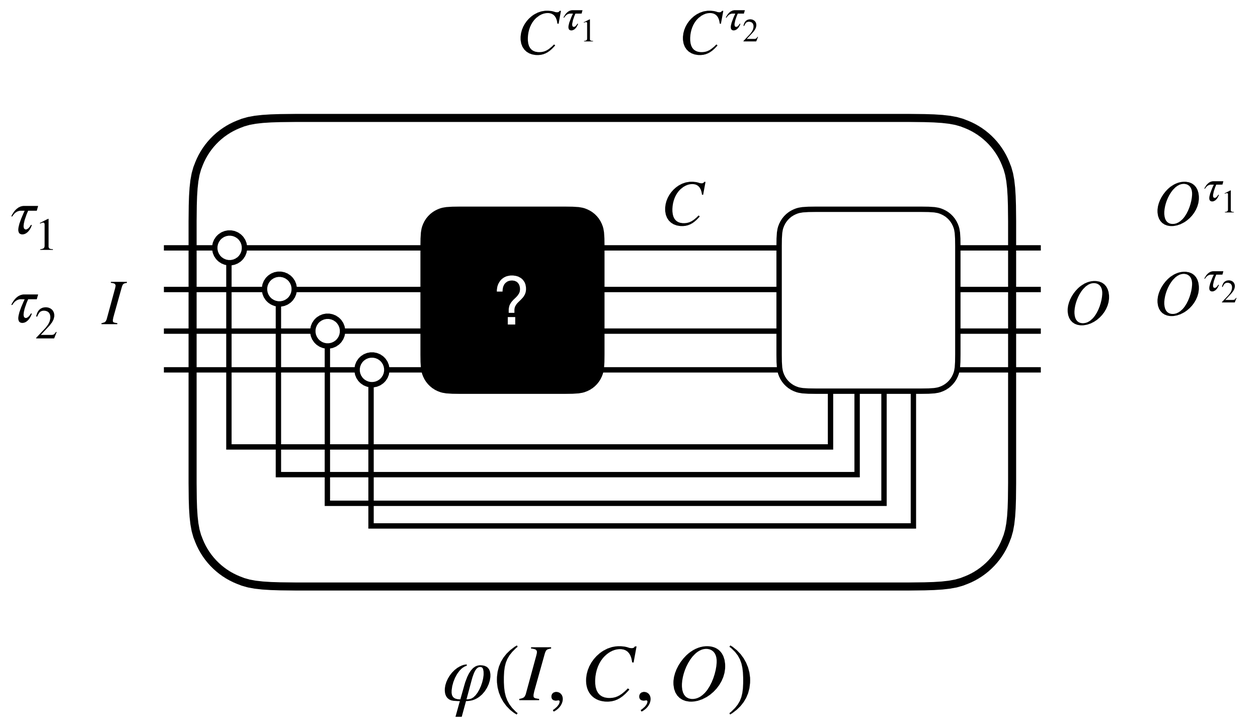
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$
$$\wedge$$

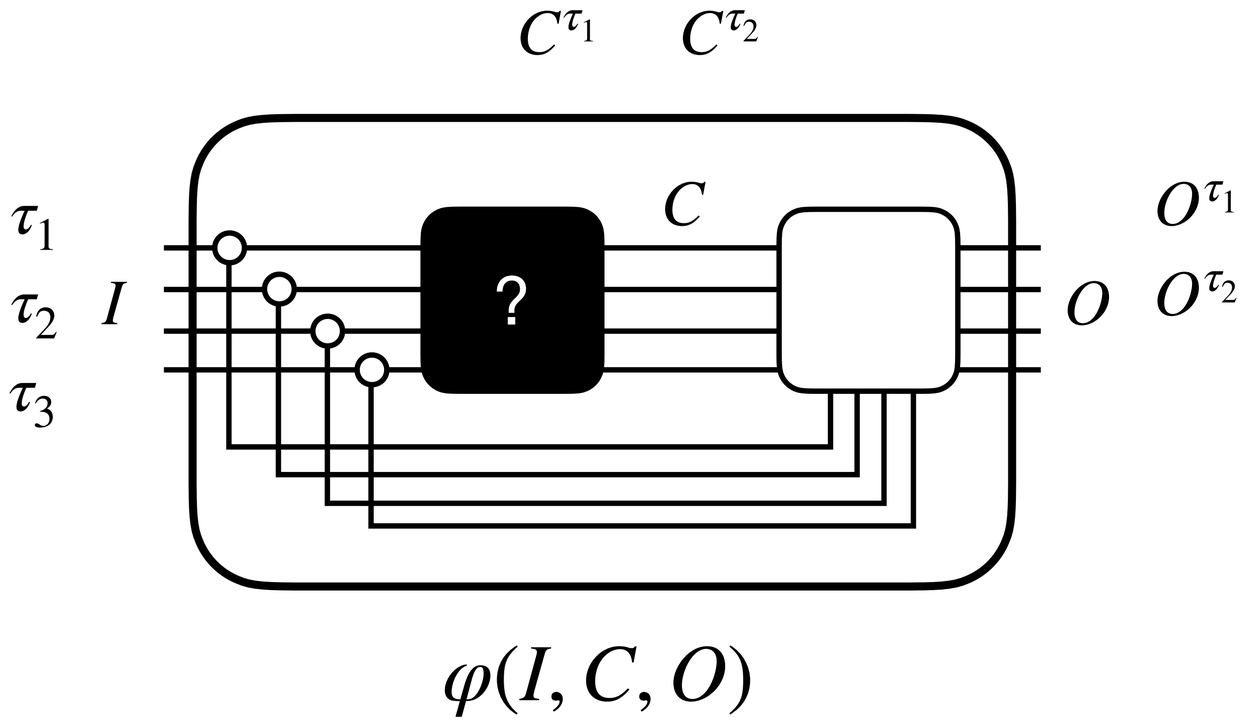
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1}) \wedge \varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$

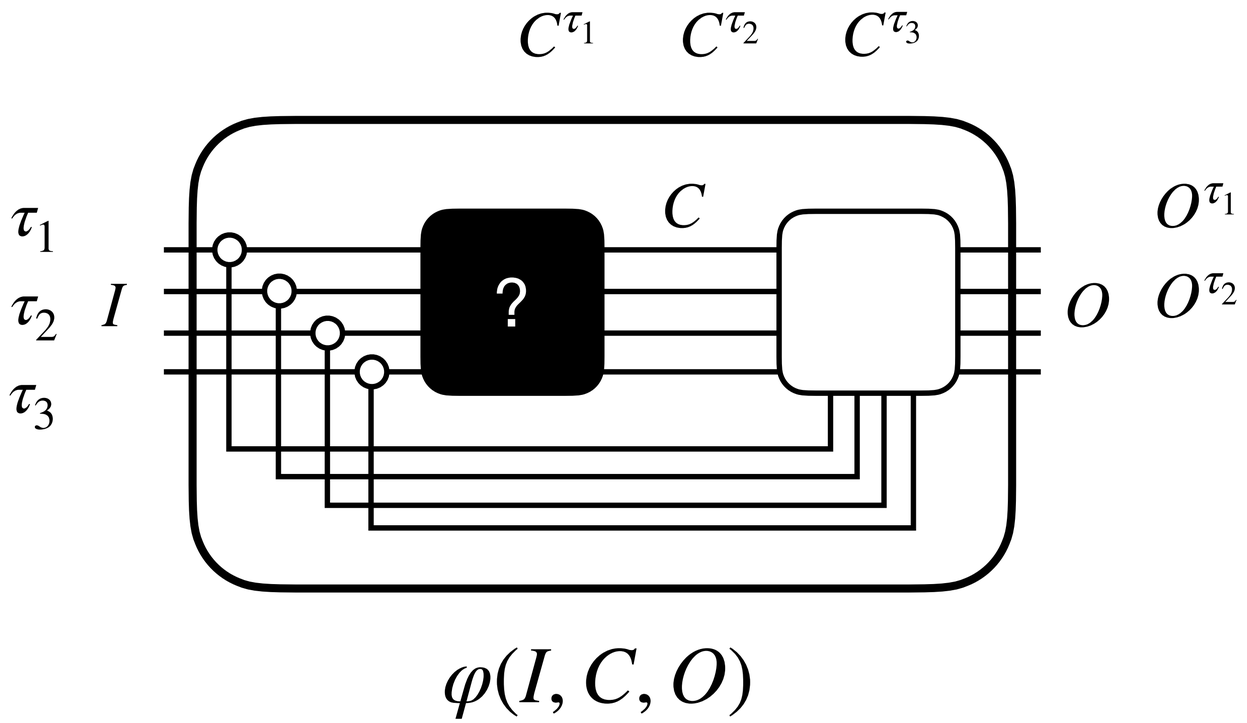
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1}) \wedge \varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$

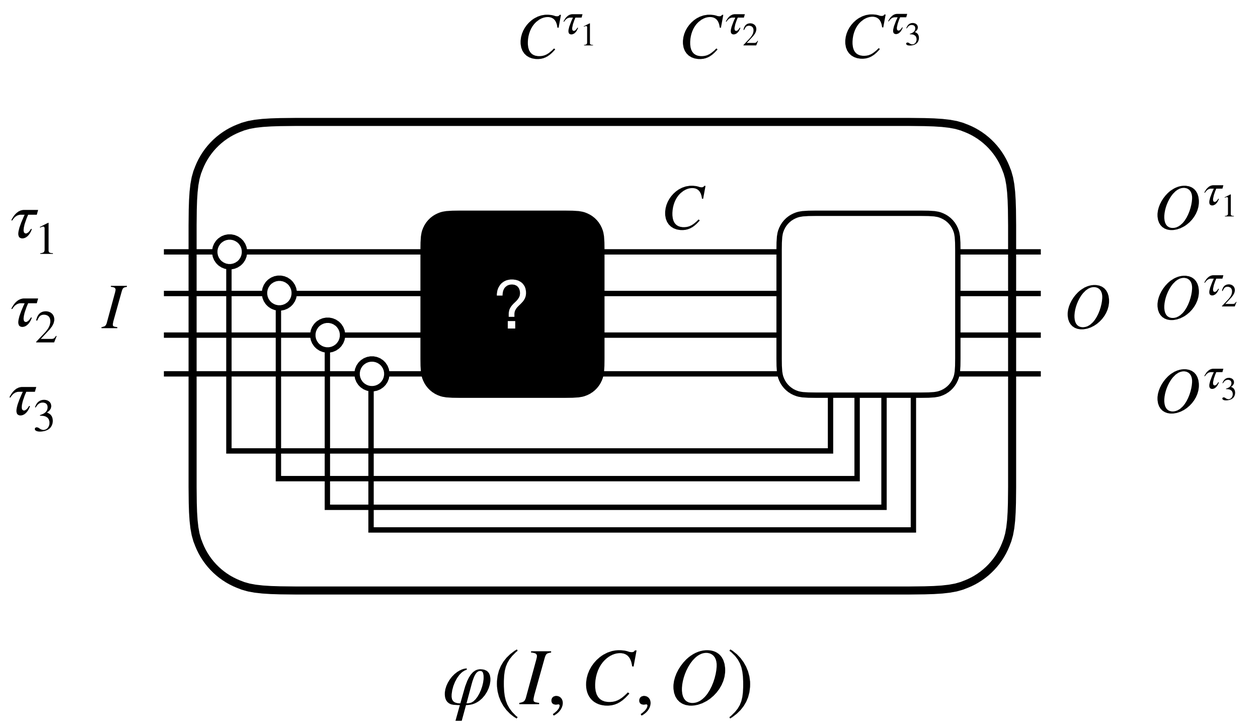
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$
$$\wedge$$
$$\varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$

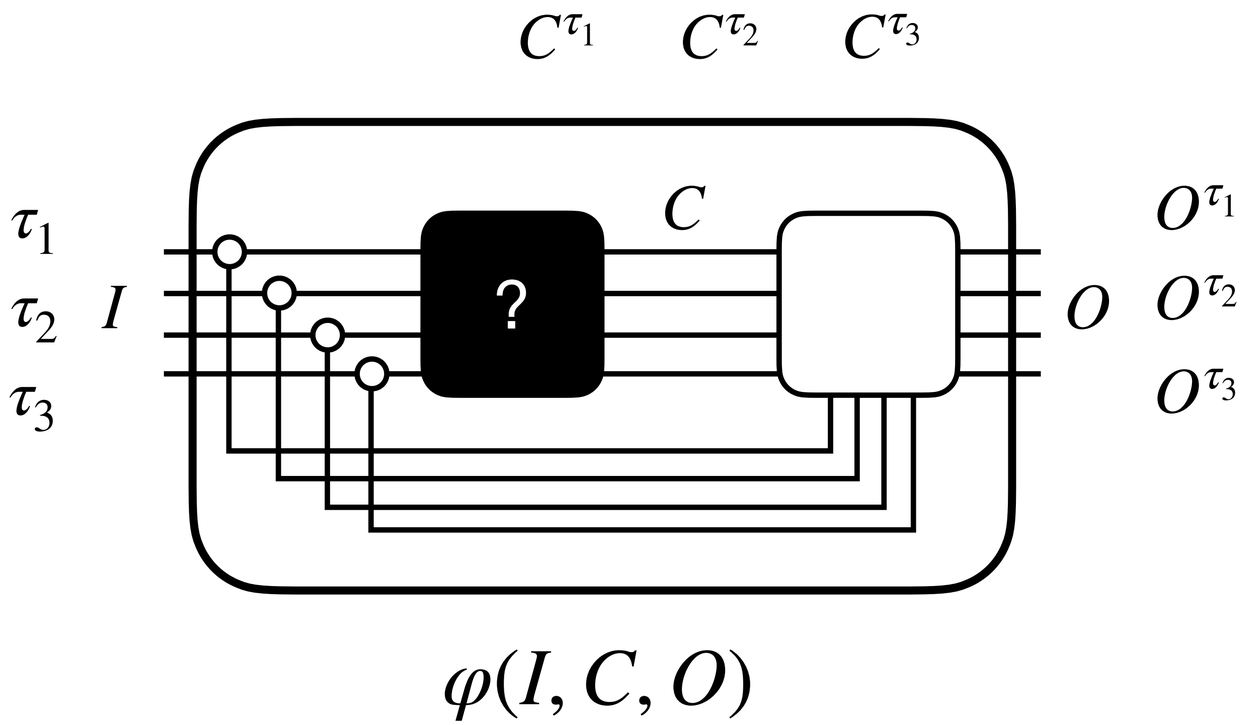
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$
$$\wedge$$
$$\varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$

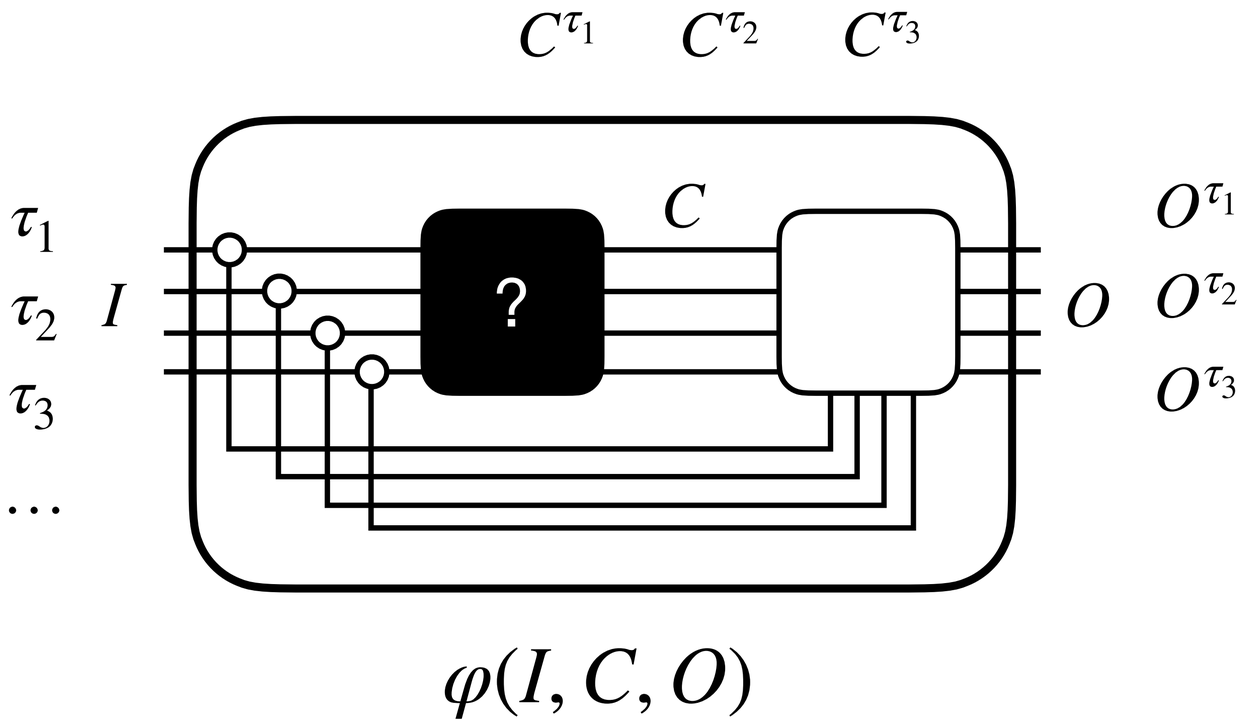
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$
$$\wedge$$
$$\varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$
$$\wedge$$

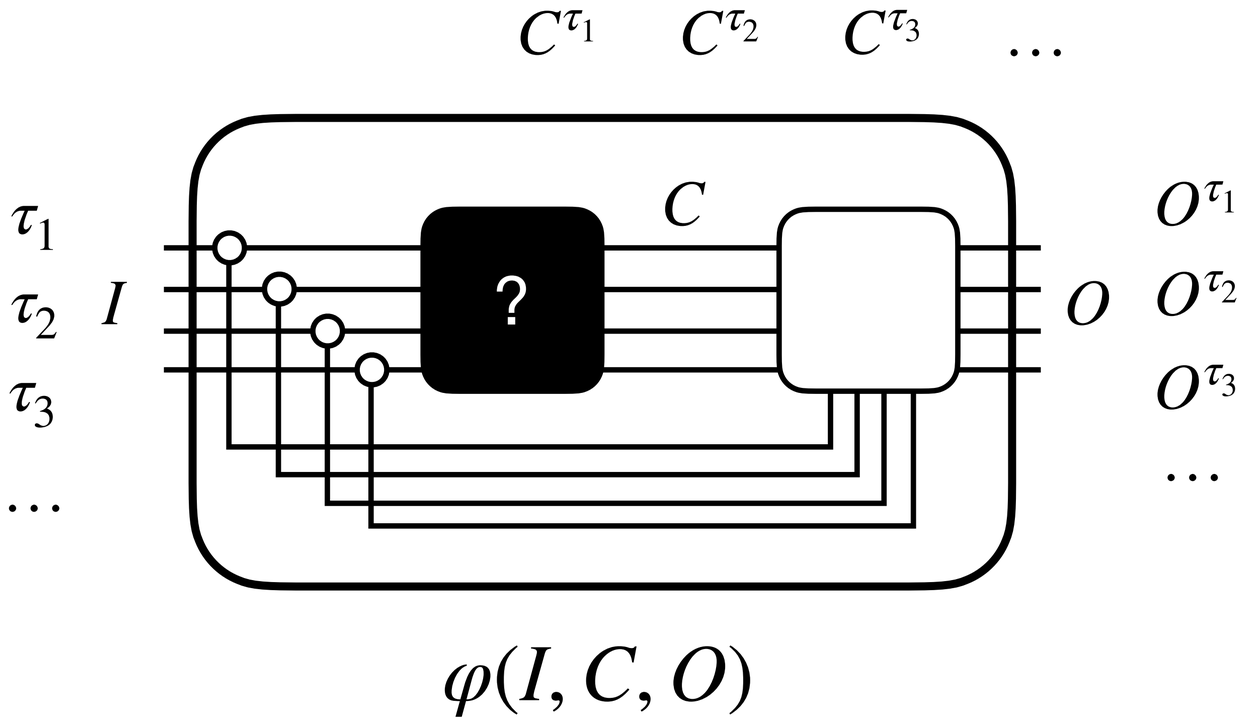
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1}) \wedge \varphi(\tau_2, C^{\tau_2}, O^{\tau_2}) \wedge \varphi(\tau_3, C^{\tau_3}, O^{\tau_3})$$

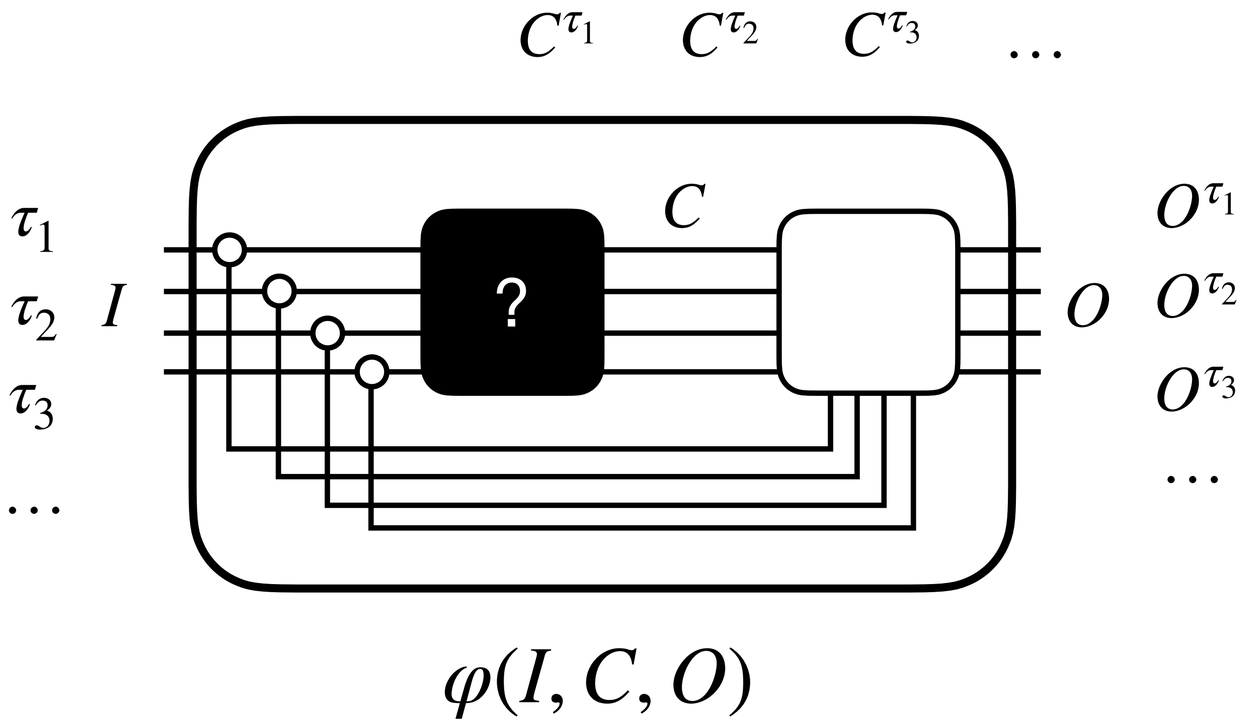
Limits of SAT



SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$
$$\wedge$$
$$\varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$
$$\wedge$$
$$\varphi(\tau_3, C^{\tau_3}, O^{\tau_3})$$
$$\wedge$$
$$\dots$$

Limits of SAT

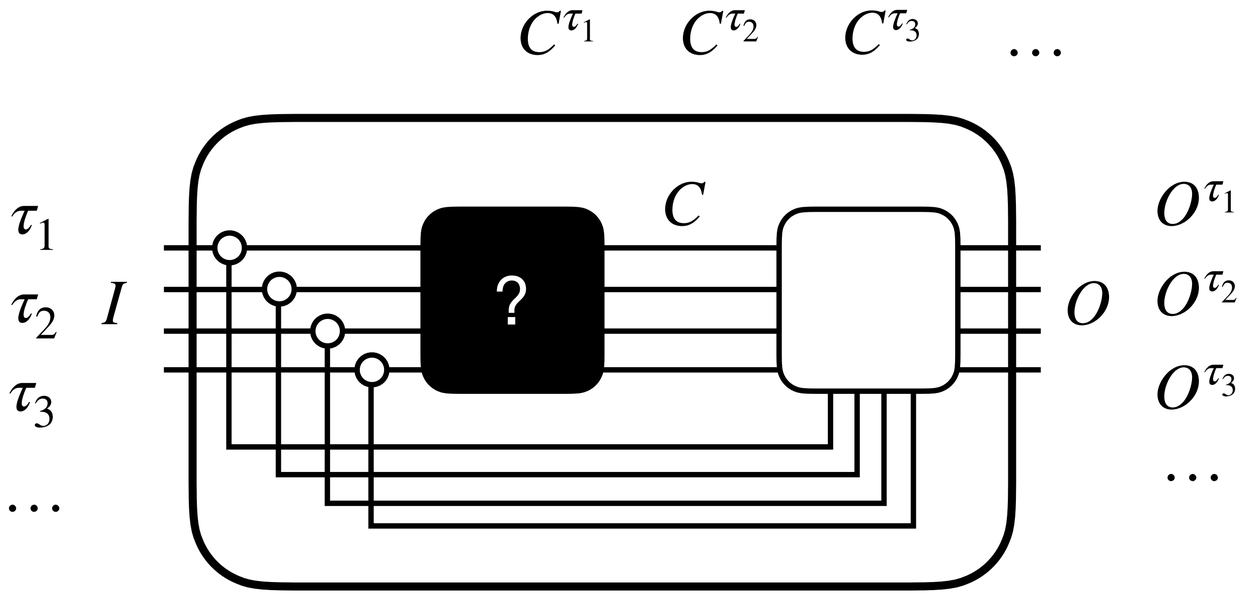


SAT encoding

$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1})$$
$$\wedge$$
$$\varphi(\tau_2, C^{\tau_2}, O^{\tau_2})$$
$$\wedge$$
$$\varphi(\tau_3, C^{\tau_3}, O^{\tau_3})$$
$$\wedge$$
$$\dots$$

$2^{|I|}$ copies!

Limits of SAT



$\forall I \exists C, O \; \varphi(I, C, O)$

SAT encoding

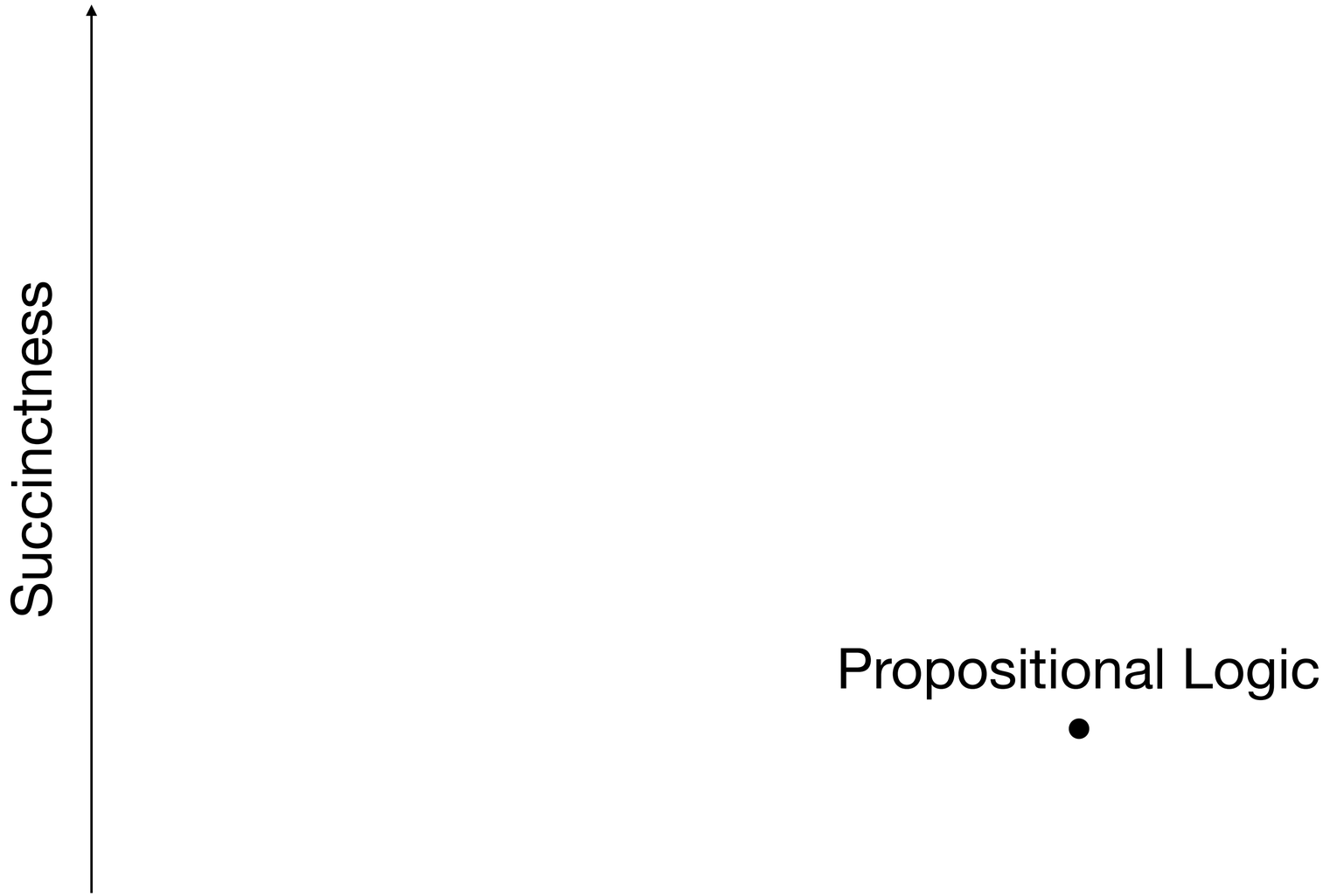
$$\varphi(\tau_1, C^{\tau_1}, O^{\tau_1}) \wedge \varphi(\tau_2, C^{\tau_2}, O^{\tau_2}) \wedge \varphi(\tau_3, C^{\tau_3}, O^{\tau_3}) \wedge \dots$$

$2^{|I|}$ copies!

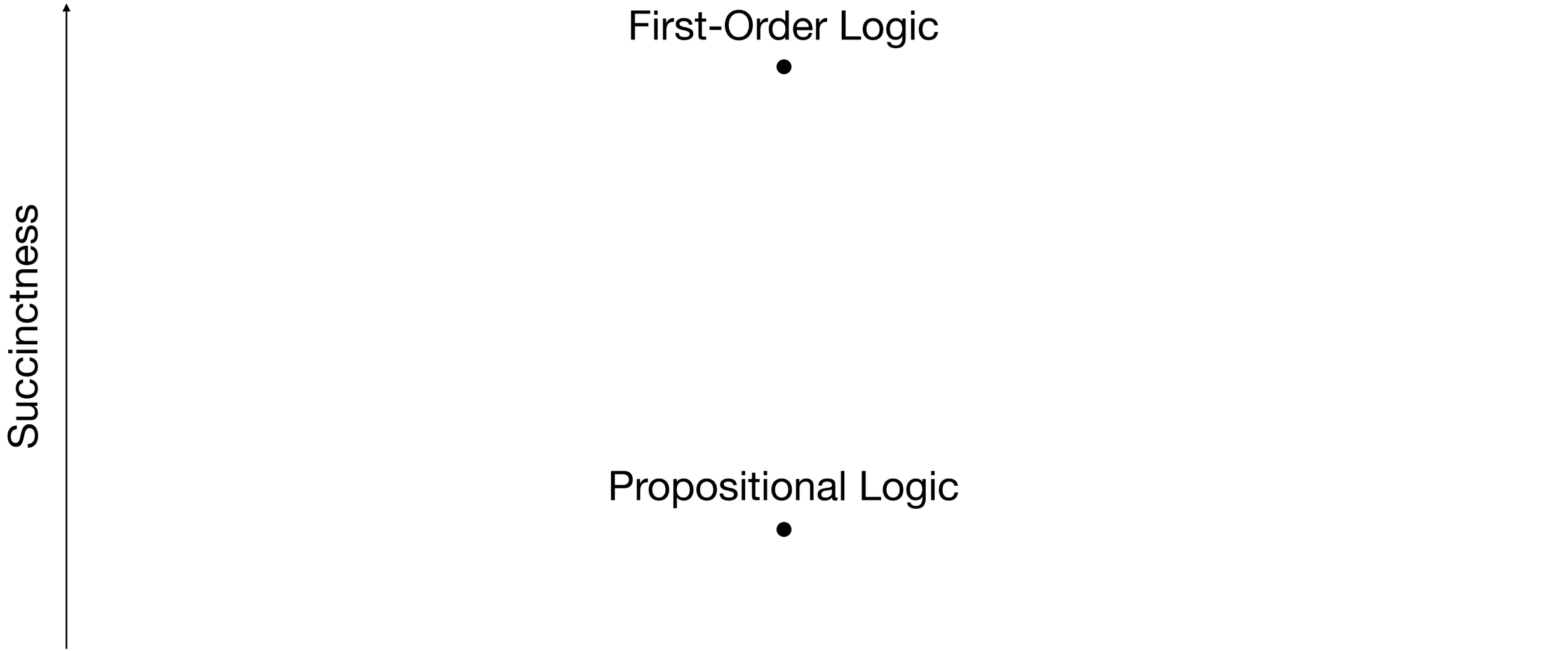
Logics for Automated Reasoning

- Propositional Logic

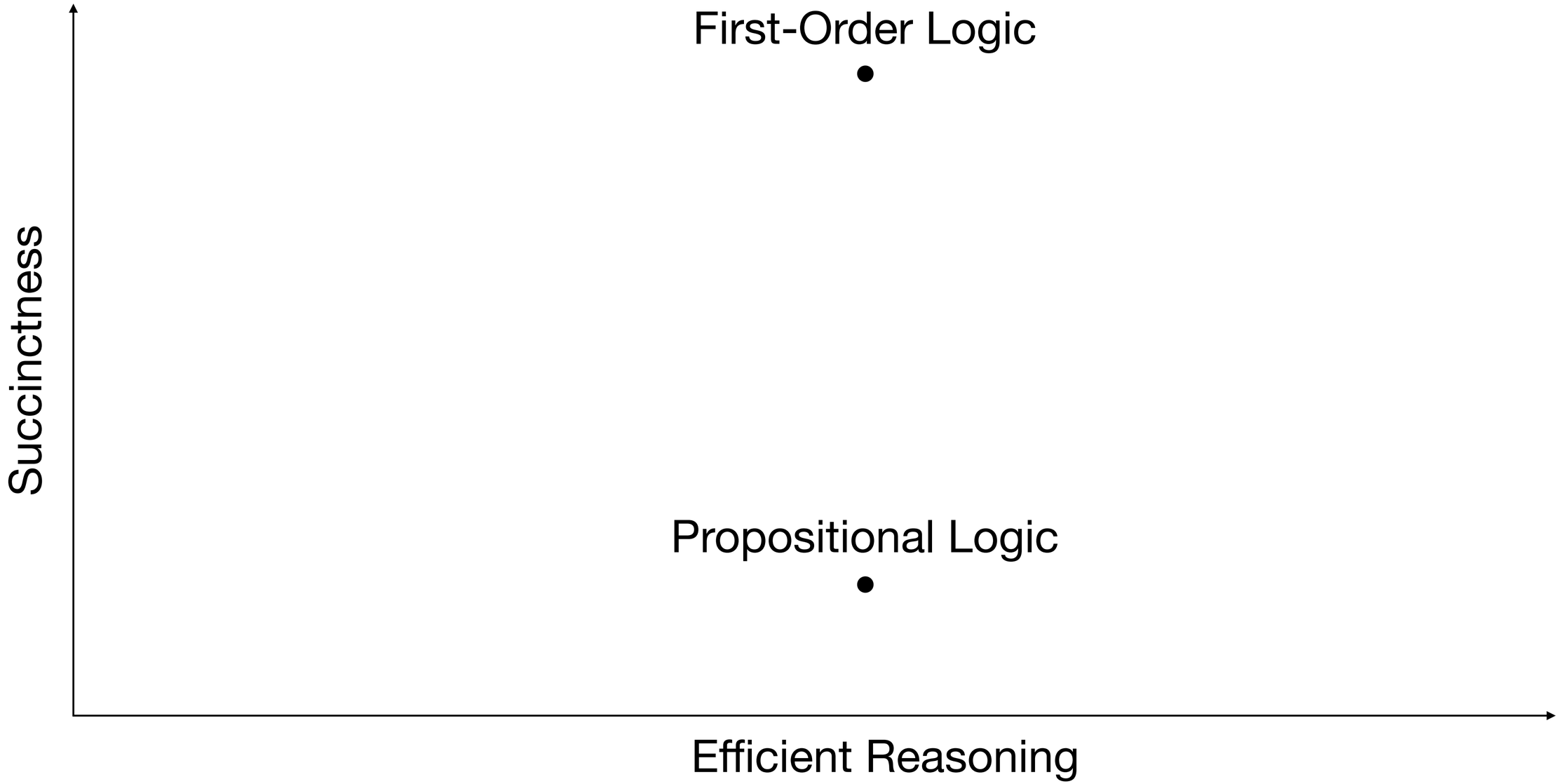
Logics for Automated Reasoning



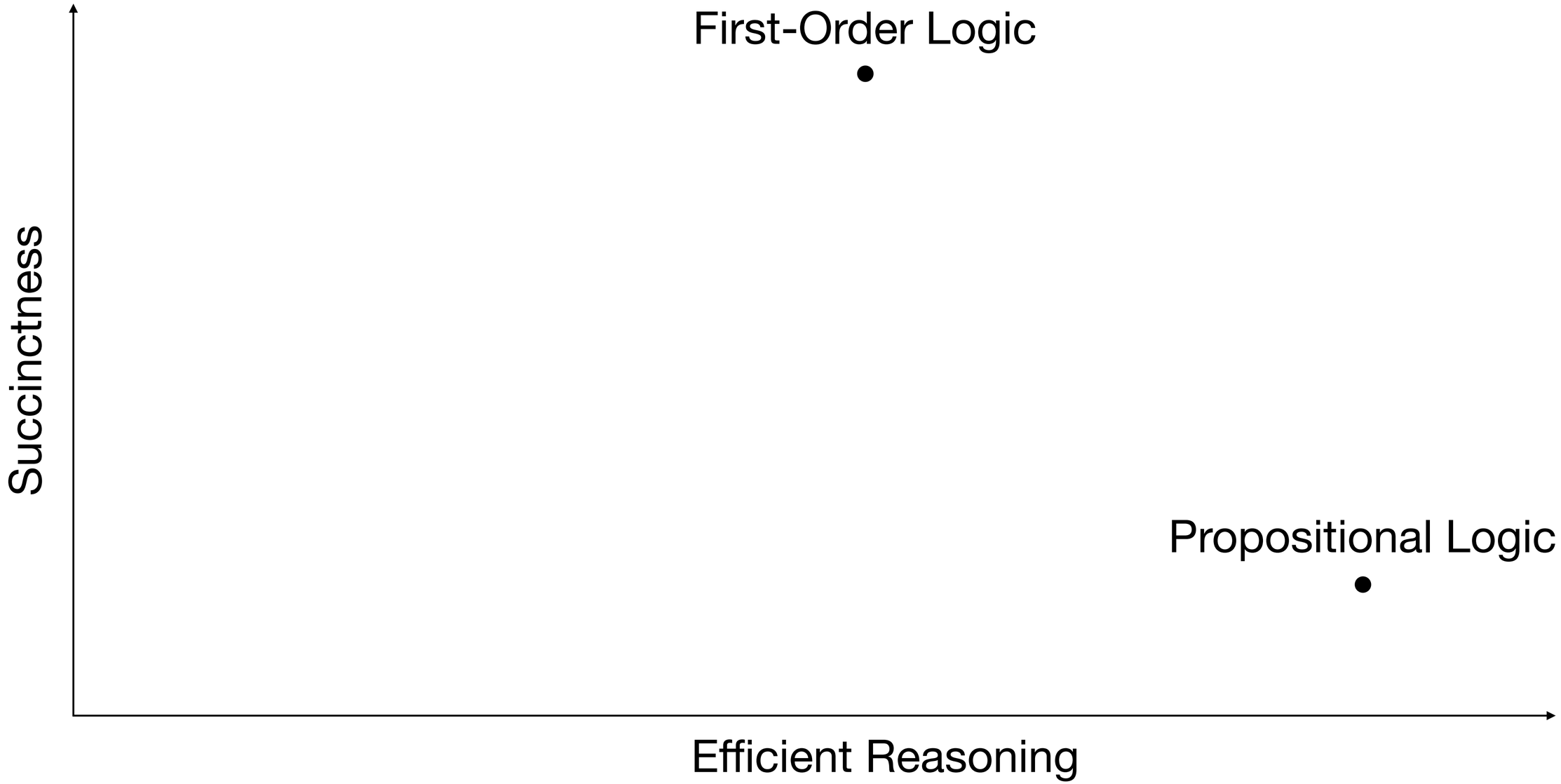
Logics for Automated Reasoning



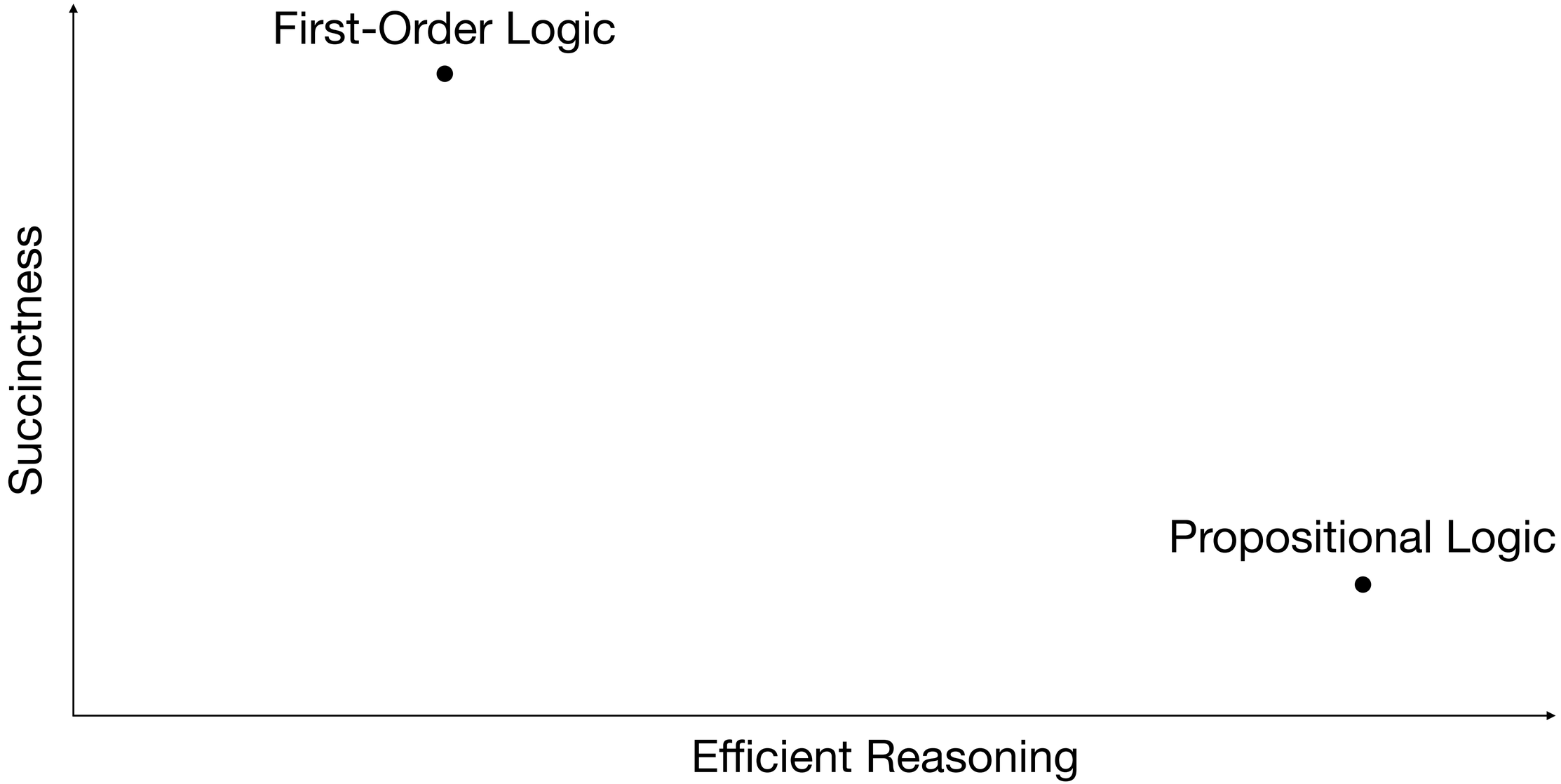
Logics for Automated Reasoning



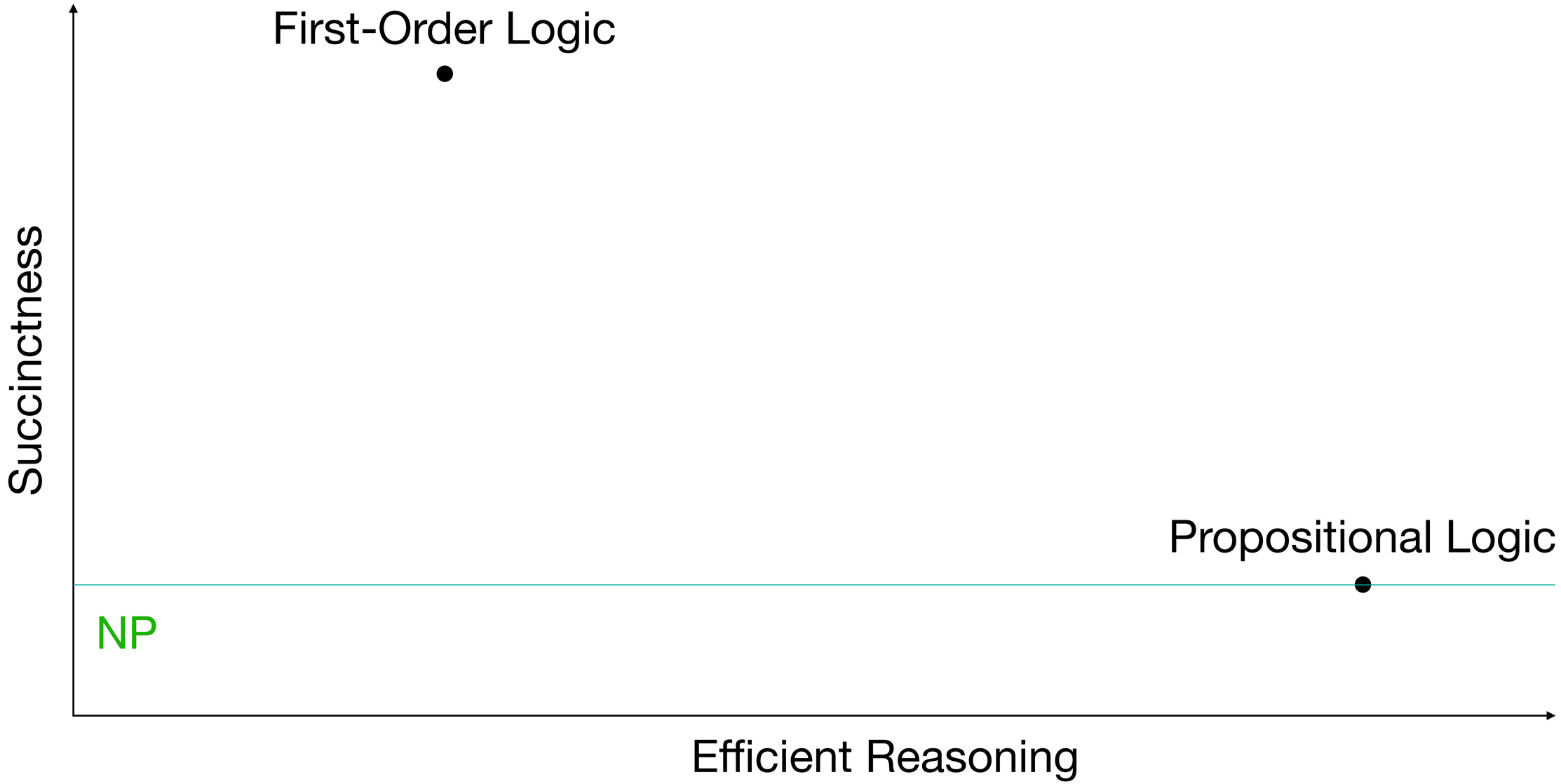
Logics for Automated Reasoning



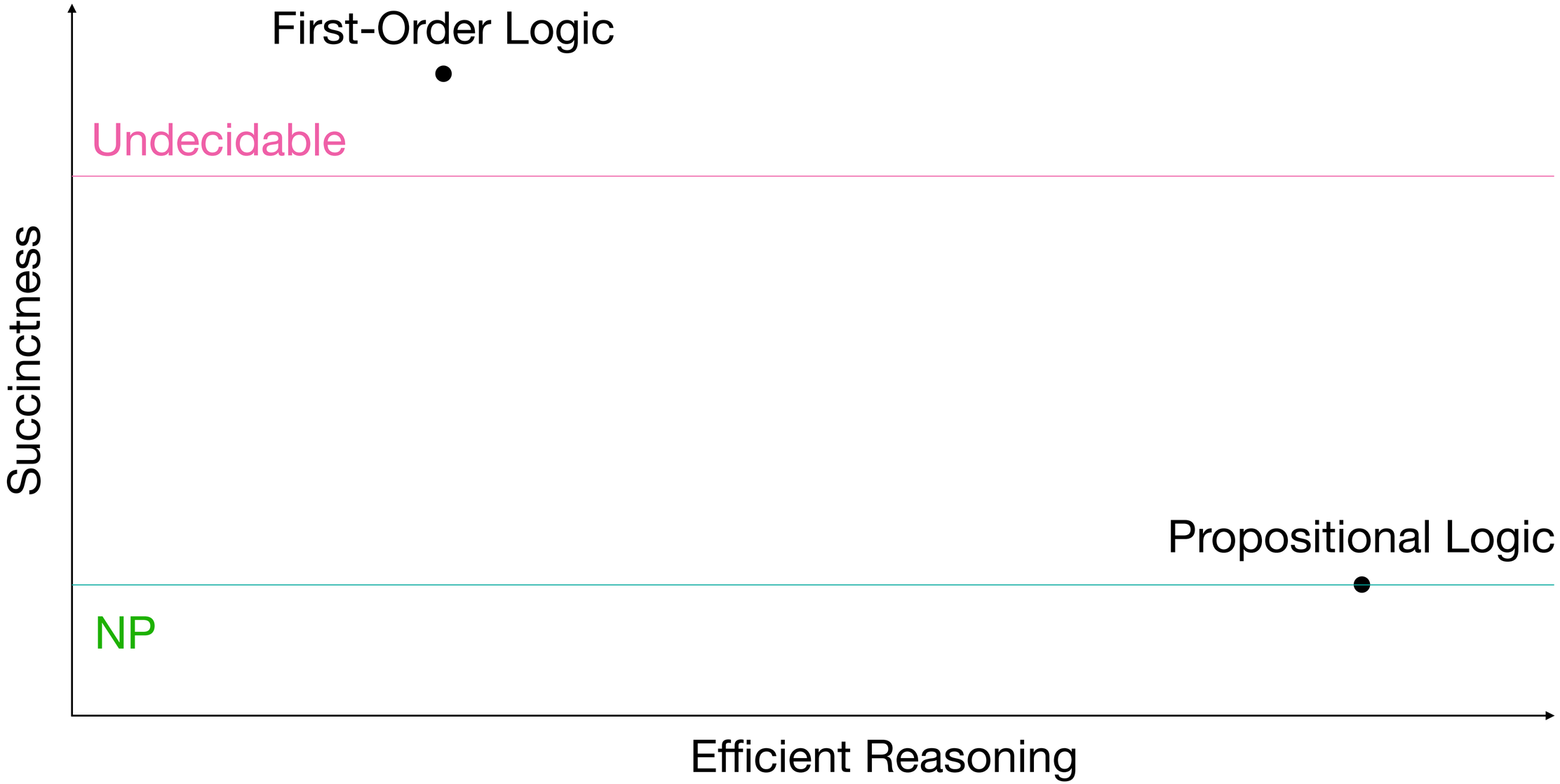
Logics for Automated Reasoning



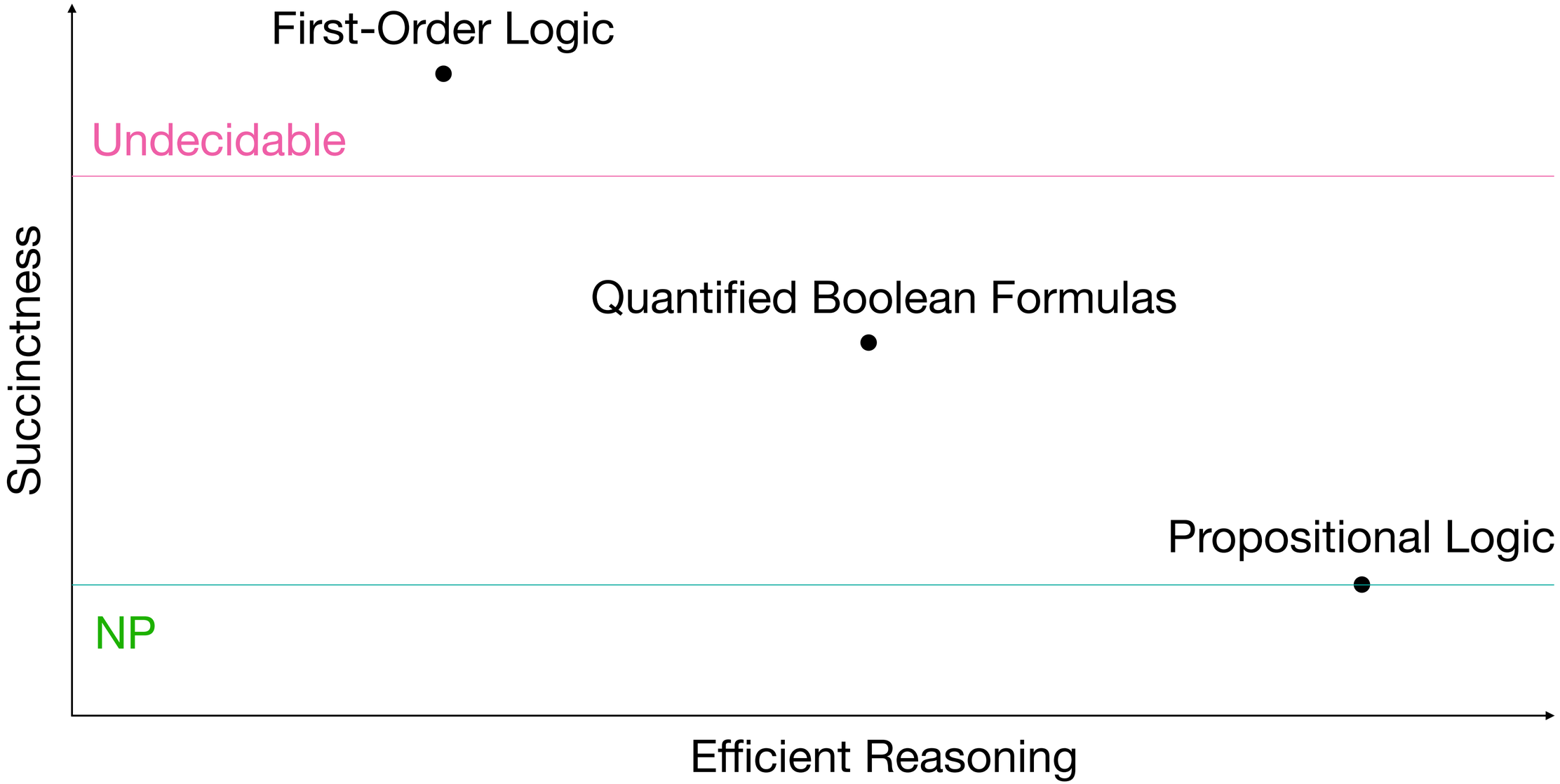
Logics for Automated Reasoning



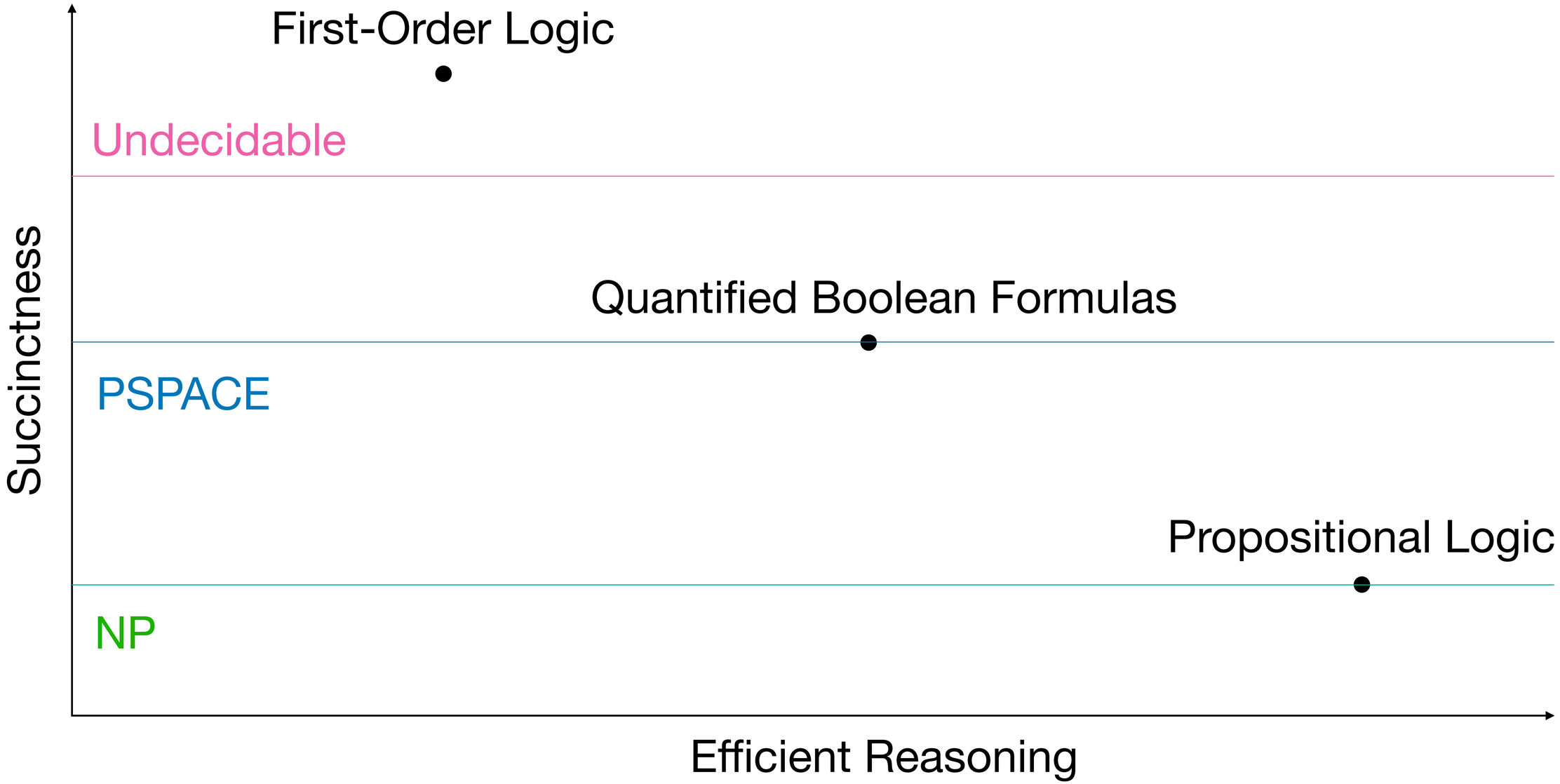
Logics for Automated Reasoning



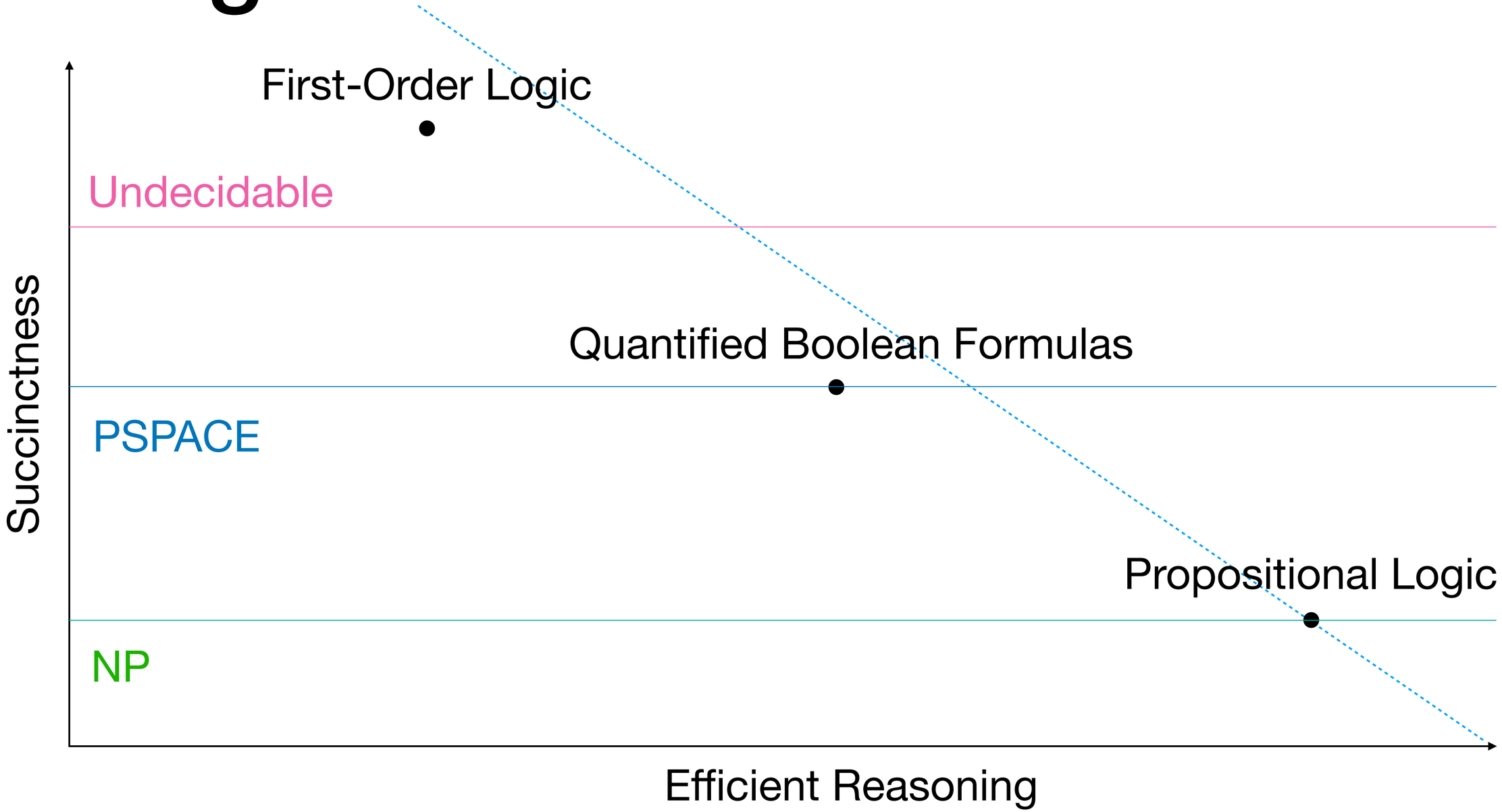
Logics for Automated Reasoning



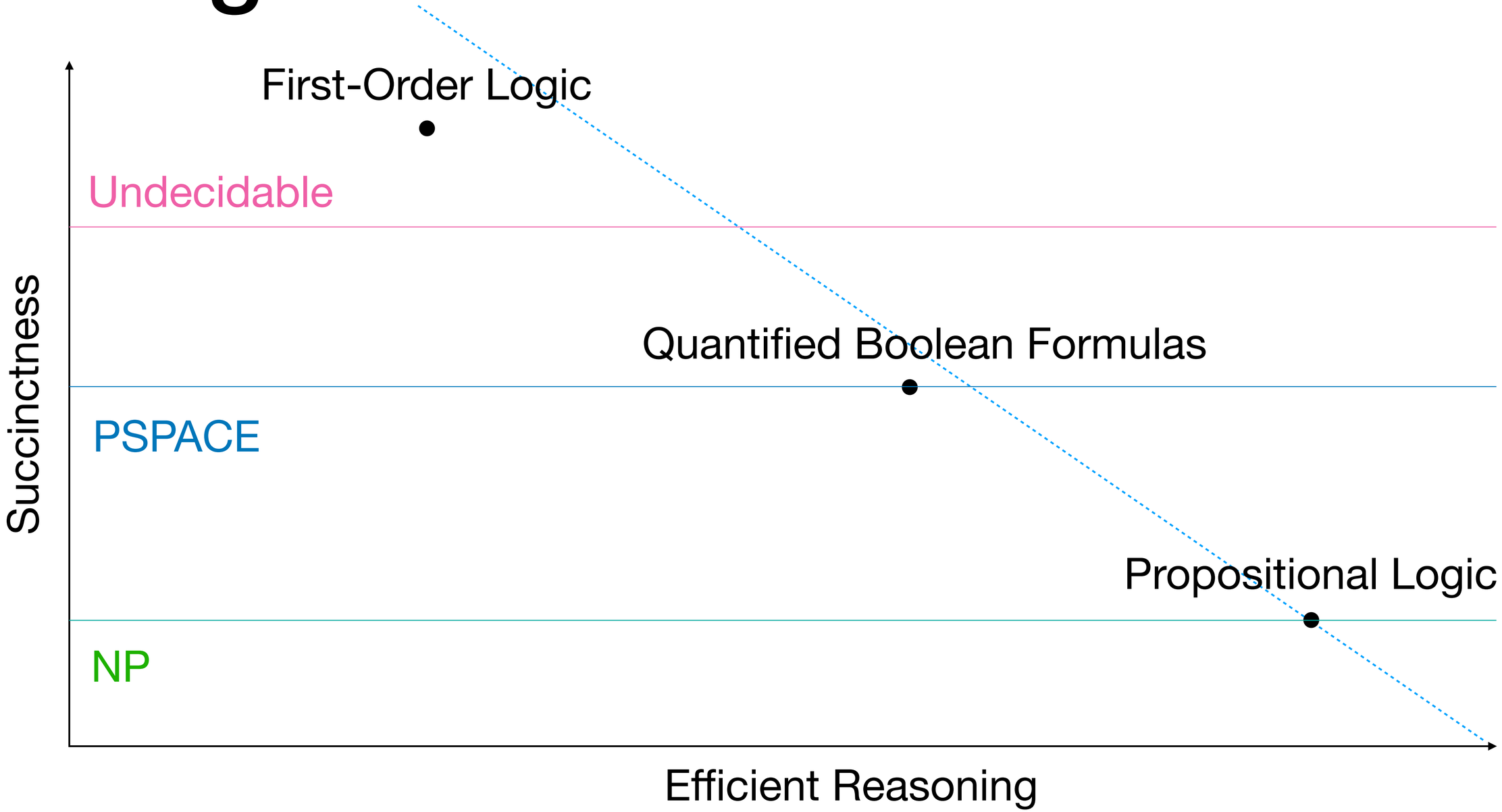
Logics for Automated Reasoning



Logics for Automated Reasoning



Logics for Automated Reasoning



Syntax, Semantics, Complexity

Syntax

Syntax

QBF ::=

Syntax

QBF ::= <Propositional Variable>

Syntax

QBF ::= <Propositional Variable> |

Syntax

QBF ::= <Propositional Variable> |

¬<QBF>

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> |

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>)

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) |

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>)

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>) |

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>) |

(∃<Propositional Variable><QBF>)

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>) |

(∃<Propositional Variable><QBF>) |

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>) |

(∃<Propositional Variable><QBF>) |

(∀<Propositional Variable><QBF>)

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>) |

(∃<Propositional Variable><QBF>) |

(∀<Propositional Variable><QBF>)

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\ x \wedge \neg u)))$$

Syntax

QBF ::= <Propositional Variable> |

¬<QBF> | (<QBF> ∨ <QBF>) | (<QBF> ∧ <QBF>) |

(∃<Propositional Variable><QBF>) |

(∀<Propositional Variable><QBF>)

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\ x \wedge \neg u)))$$

$$\forall x \forall y \exists z(z \vee \neg(x \wedge y)) \wedge (\neg z \vee (x \wedge y))$$

Semantics

Semantics

$\forall x \Phi$

Semantics

$\forall x \Phi \quad \equiv$

Semantics

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

Semantics

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

Substitute $\top$ for every **free** occurrence of x.

Semantics

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi$$

Substitute $\top$ for every **free** occurrence of x.

Semantics

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv$$

Substitute $\top$ for every **free** occurrence of x.

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\, x \wedge \neg u)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\, x \wedge \neg u))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((x \wedge \neg \top) \vee (x \wedge \neg \bot)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\ ] \wedge \Phi[x \leftarrow \top\ ]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\ ] \vee \Phi[x \leftarrow \top\ ]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\ x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\ x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\ x \wedge \neg u))) \equiv$$

$$(\forall x((x \wedge \neg \top) \vee (x \wedge \neg \bot)))$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((x \wedge \neg \top) \vee (x \wedge \neg \bot))) \equiv$$

$$(\forall x\, x)$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((x \wedge \neg \top) \vee (x \wedge \neg \bot))) \equiv$$

$$(\forall x\, x)$$

Semantics

Substitute $\top$ for every **free** occurrence of x.

$$\forall x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \wedge \Phi[x \leftarrow \top\,]$$

$$\exists x \Phi \quad \equiv \quad \Phi[x \leftarrow \bot\,] \vee \Phi[x \leftarrow \top\,]$$

$$(\forall x((\exists y \neg(y \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((\neg(\bot \wedge z) \vee (\neg \top \wedge z)) \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x \top \wedge (\exists u\, x \wedge \neg u))) \equiv$$

$$(\forall x((x \wedge \neg \top) \vee (x \wedge \neg \bot))) \equiv$$

$$(\forall x\, x) \equiv$$

$$\bot$$

Prenex Normal Form (PNF)

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$$(\exists x \Phi) \circ \Psi$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \quad \Longleftrightarrow$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \quad \Longleftrightarrow \quad (\exists x \Phi \circ \Psi)^{*}$$

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \quad \Longleftrightarrow \quad (\exists x \Phi \circ \Psi)^{*}$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^{*}$$

$$(\forall x \Phi) \circ \Psi$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^{*}$$

$$(\forall x \Phi) \circ \Psi \iff$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^{*}$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^{*}$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^{*}$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^*$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^*$$

$$\neg(\forall x \Phi)$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^{*}$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^{*}$$

$$\neg(\forall x \Phi) \iff$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^*$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^*$$

$$\neg(\forall x \Phi) \iff \exists x \Phi$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^*$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^*$$

$$\neg(\forall x \Phi) \iff \exists x \Phi$$

$$\neg(\exists x \Phi)$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^{*}$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^{*}$$

$$\neg(\forall x \Phi) \iff \exists x \Phi$$

$$\neg(\exists x \Phi) \iff$$

*Variable x must not occur free in Ψ (rename if necessary).

Prenex Normal Form (PNF)

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$\circ \in \{ \wedge, \vee \}$

$$(\exists x \Phi) \circ \Psi \iff (\exists x \Phi \circ \Psi)^*$$

$$(\forall x \Phi) \circ \Psi \iff (\forall x \Phi \circ \Psi)^*$$

$$\neg(\forall x \Phi) \iff \exists x \Phi$$

$$\neg(\exists x \Phi) \iff \forall x \Phi$$

*Variable x must not occur free in Ψ (rename if necessary).

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
  if Q == []:
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
  if Q == []:
    return simplify(φ)
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
  if Q == []:
    return simplify(φ)
  elif Q_1 == ∃:
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
  if Q == []:
    return simplify(φ)
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
  if Q == []:
    return simplify(φ)
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

```
def evaluate(Q, φ):
  if Q == []:
    return simplify(φ)
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

A Simple Recursive Algorithm

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$var(\varphi) = \{x_1, \ldots, x_n\}$

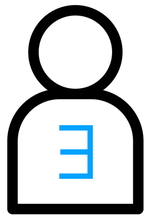
```
def evaluate(Q, φ):
  if Q == []:
    return simplify(φ)
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

QSAT is in **PSPACE**

QBF Game Semantics

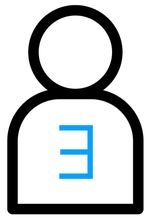
$$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$$

QBF Game Semantics

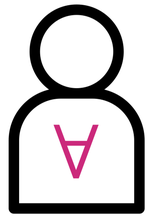


$$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$$

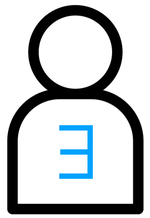
QBF Game Semantics



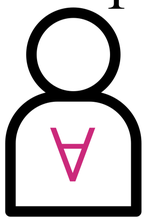
$$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$$



QBF Game Semantics



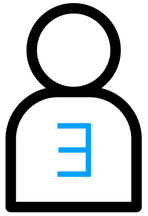
$$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$$



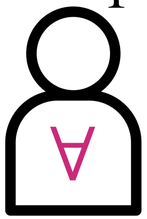
$$U_1 = \tau_1$$

QBF Game Semantics

$E_1 = \sigma_1$



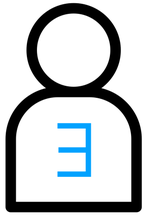
$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$



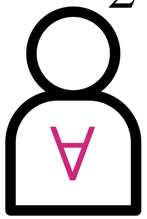
$U_1 = \tau_1$

QBF Game Semantics

$E_1 = \sigma_1$



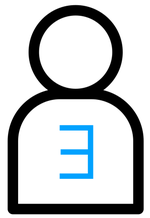
$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$



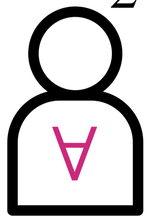
$U_1 = \tau_1 \quad U_2 = \tau_2$

QBF Game Semantics

$E_1 = \sigma_1 \quad E_2 = \sigma_2$



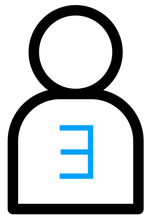
$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$



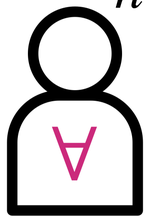
$U_1 = \tau_1 \quad U_2 = \tau_2$

QBF Game Semantics

$E_1 = \sigma_1 \quad E_2 = \sigma_2$



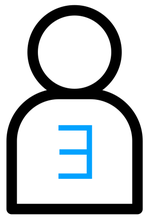
$$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$$



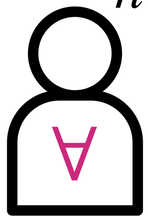
$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

QBF Game Semantics

$$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$$



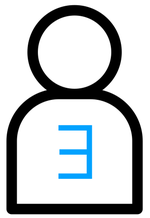
$$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$$



$$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$$

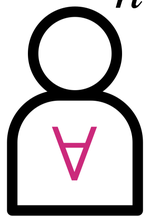
QBF Game Semantics

$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$

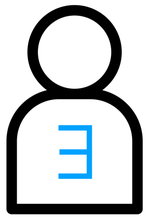
φ satisfied?



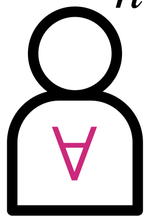
$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

QBF Game Semantics

$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$



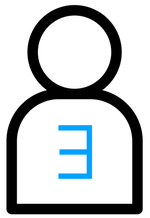
$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

φ satisfied?

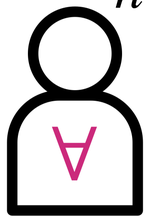
∃ wins

QBF Game Semantics

$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$



$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

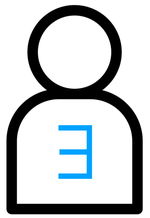
φ satisfied?

∃ wins

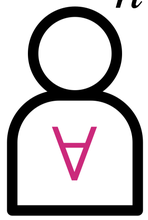
QBF is **true**

QBF Game Semantics

$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$



$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

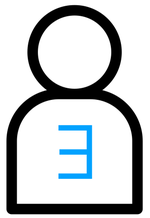
φ satisfied?

∃ wins

QBF is **true** ⟺

QBF Game Semantics

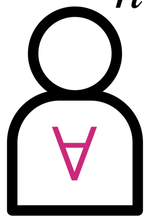
$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$

φ satisfied?

∃ wins

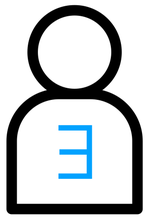


$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

QBF is **true** ⟺ ∃ has a winning strategy

QBF Game Semantics

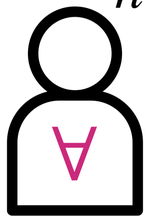
$E_1 = \sigma_1 \quad E_2 = \sigma_2 \qquad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$

φ satisfied?

∃ wins



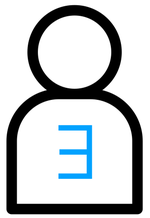
$U_1 = \tau_1 \quad U_2 = \tau_2 \qquad U_n = \tau_n$

QBF is **true** ⟺ ∃ has a winning strategy

QBF is **false**

QBF Game Semantics

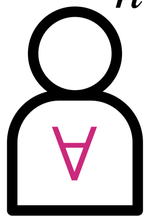
$E_1 = \sigma_1 \quad E_2 = \sigma_2 \quad E_n = \sigma_n$



$\forall U_1 \exists E_1 \forall U_2 \exists E_2 \ldots \forall U_n \exists E_n \,.\, \varphi$

φ satisfied?

∃ wins



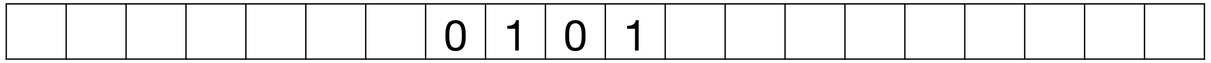
$U_1 = \tau_1 \quad U_2 = \tau_2 \quad U_n = \tau_n$

QBF is **true** ⟺ ∃ has a winning strategy

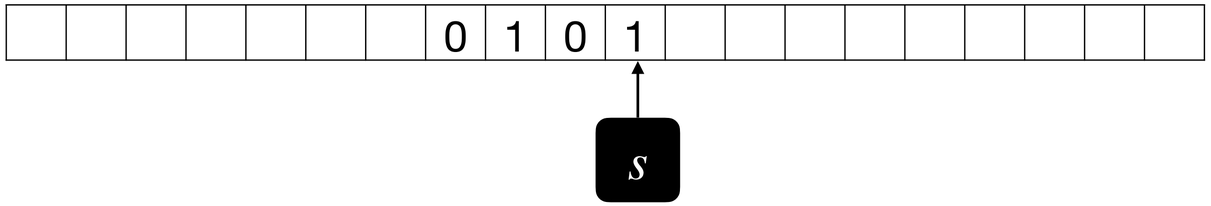
QBF is **false** ∀ has a winning strategy

Complexity of QBF Evaluation

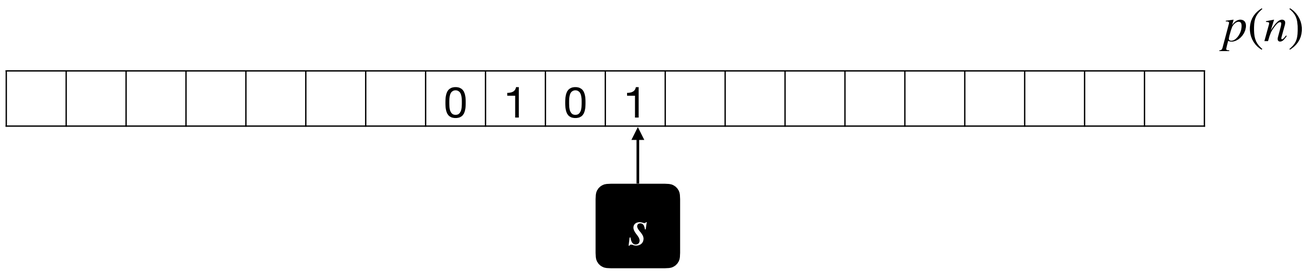
Complexity of QBF Evaluation



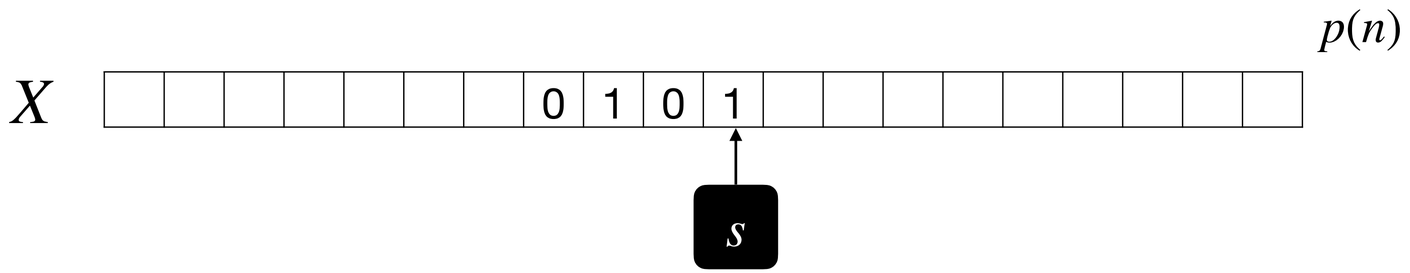
Complexity of QBF Evaluation



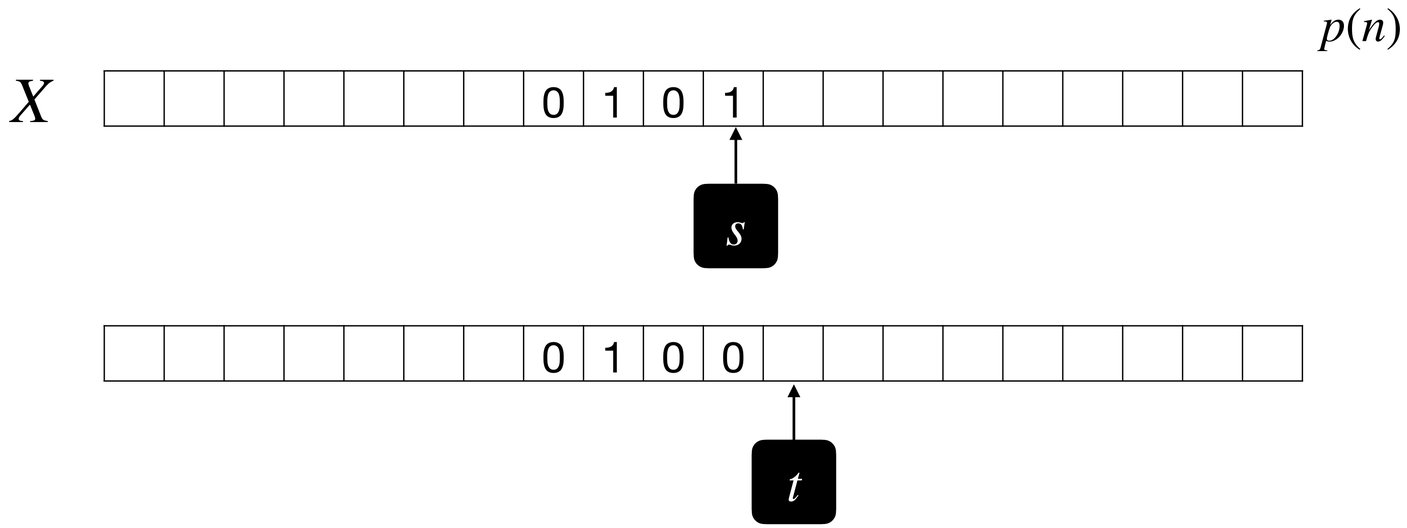
Complexity of QBF Evaluation



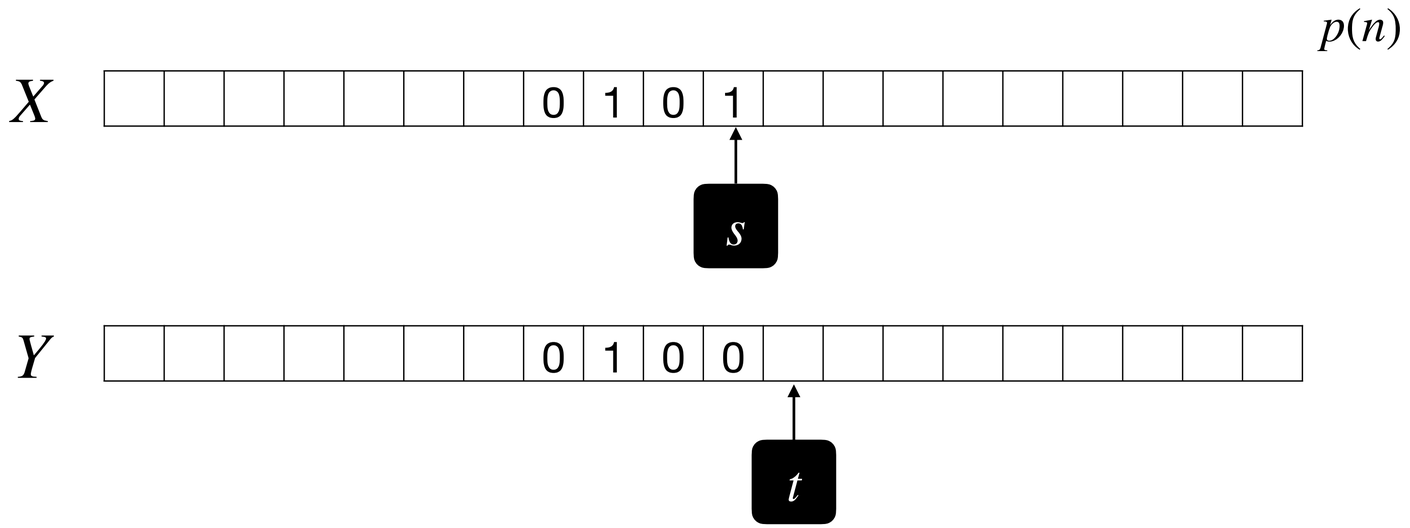
Complexity of QBF Evaluation



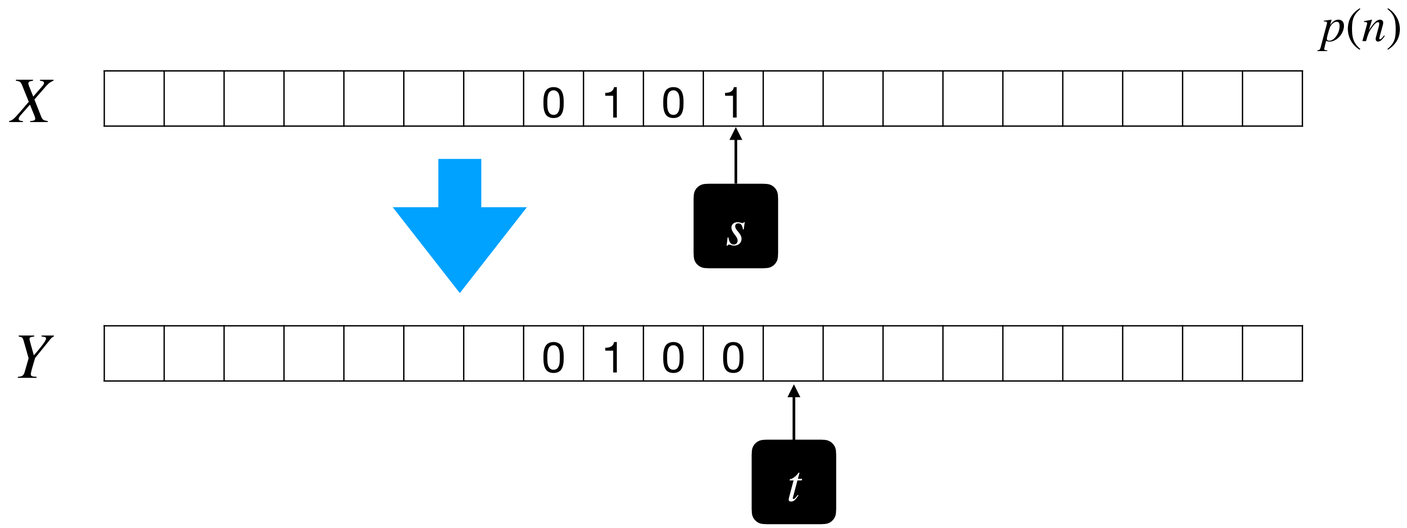
Complexity of QBF Evaluation



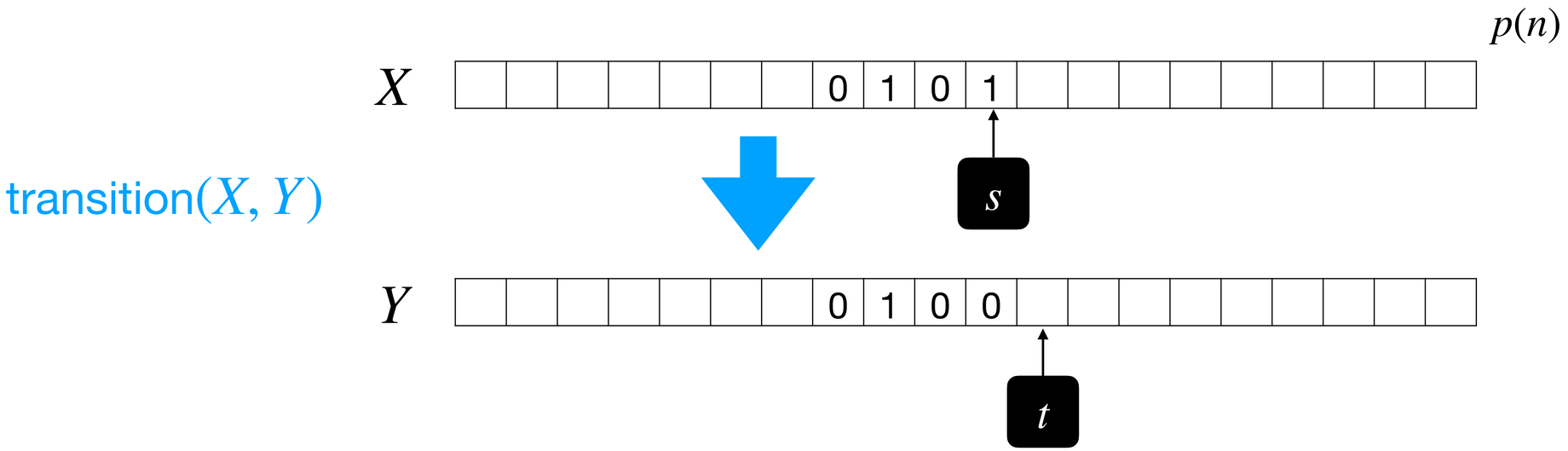
Complexity of QBF Evaluation



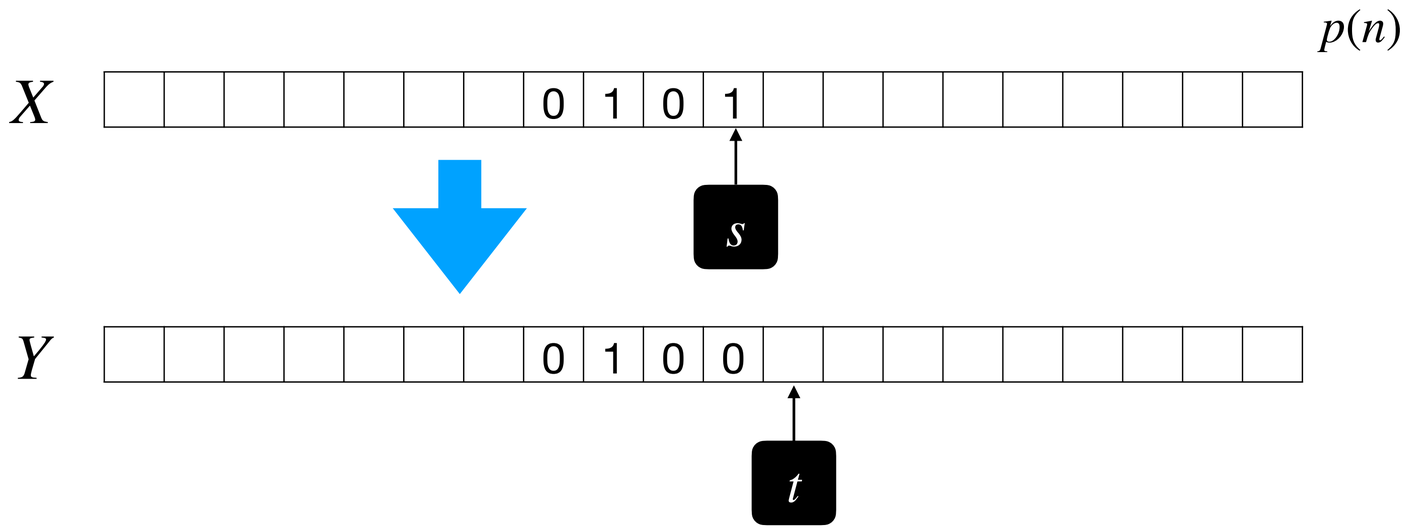
Complexity of QBF Evaluation



Complexity of QBF Evaluation



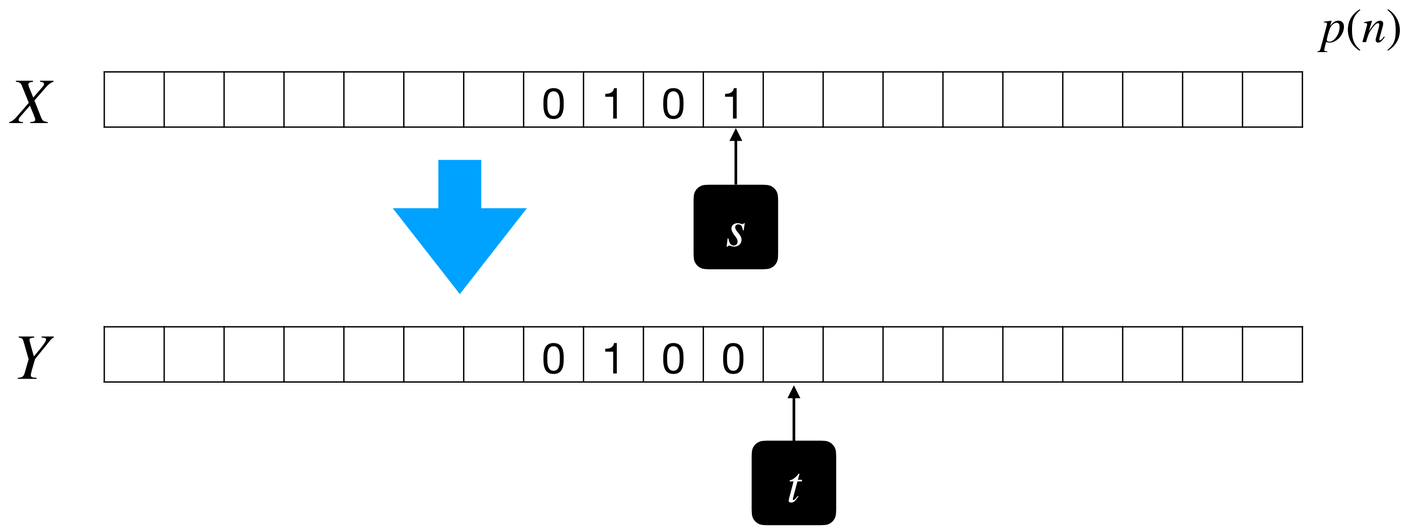
Complexity of QBF Evaluation



transition(X, Y)

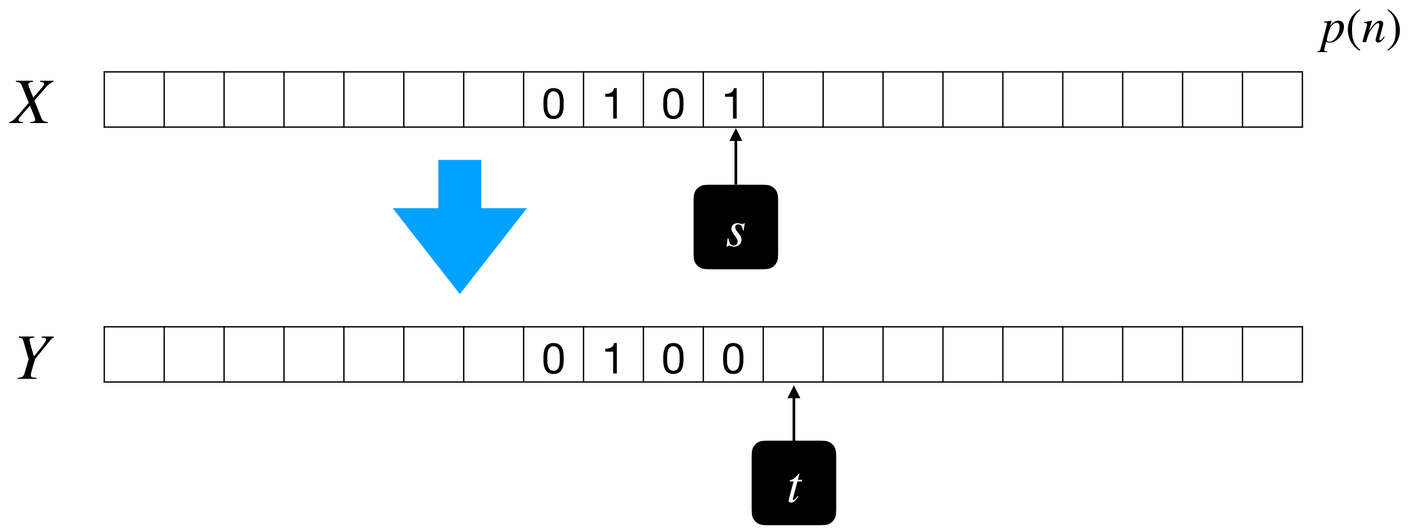
reach$_{\leq 1}(X, Y)$

Complexity of QBF Evaluation



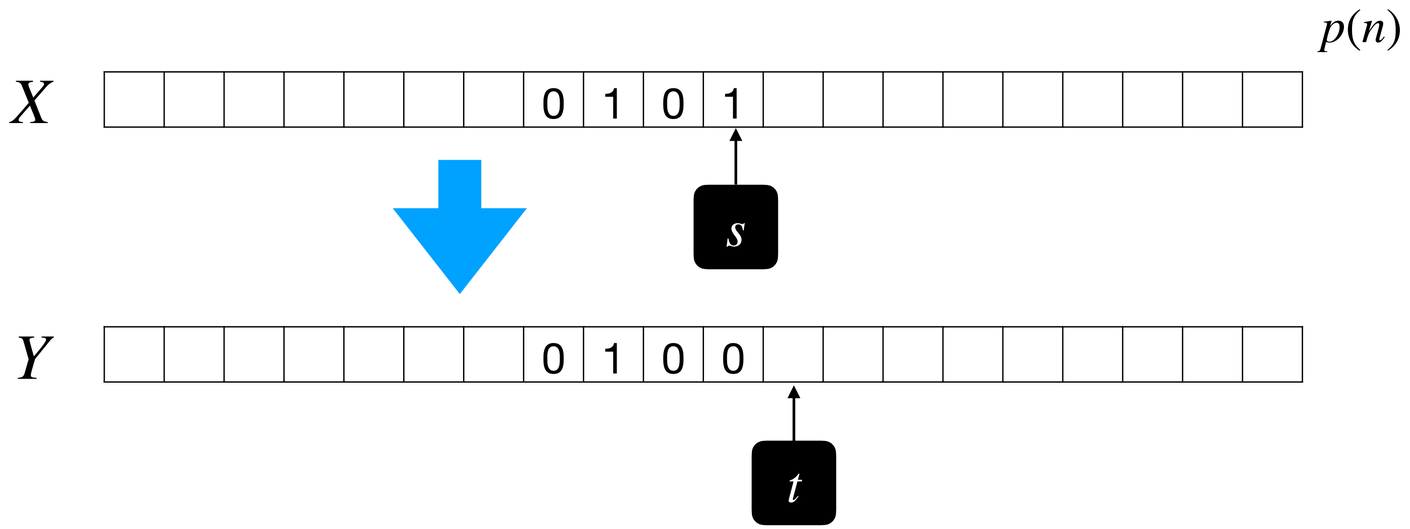
reach$_{\leq 1}(X, Y)$:=

Complexity of QBF Evaluation



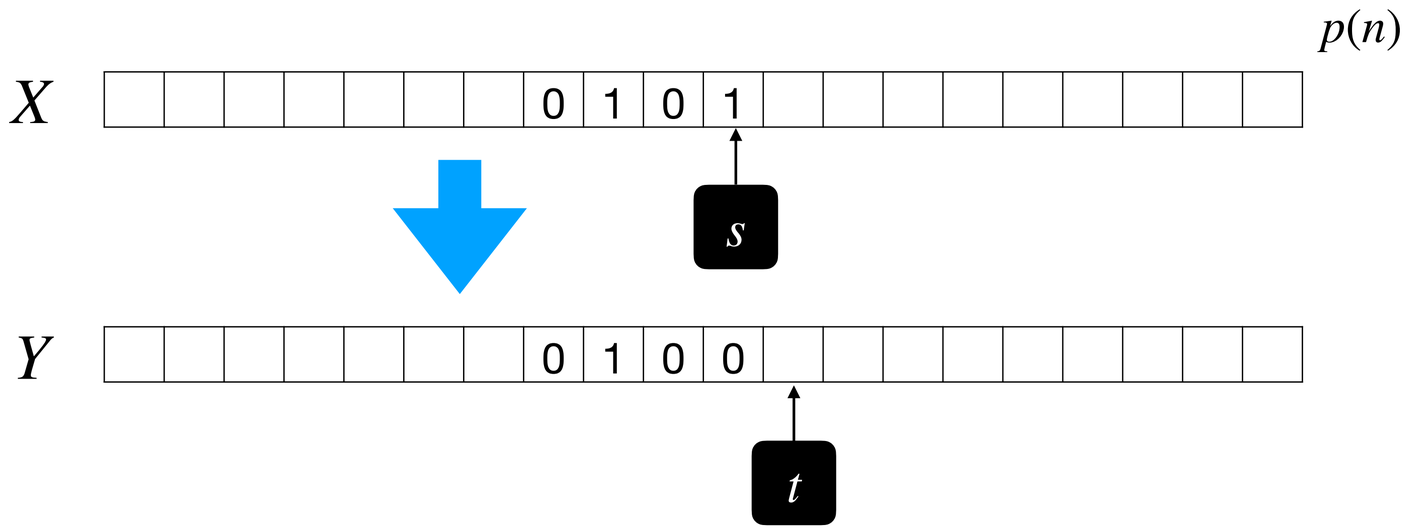
reach$_{\leq 1}(X, Y)$ $:=$ $X \leftrightarrow Y$

Complexity of QBF Evaluation



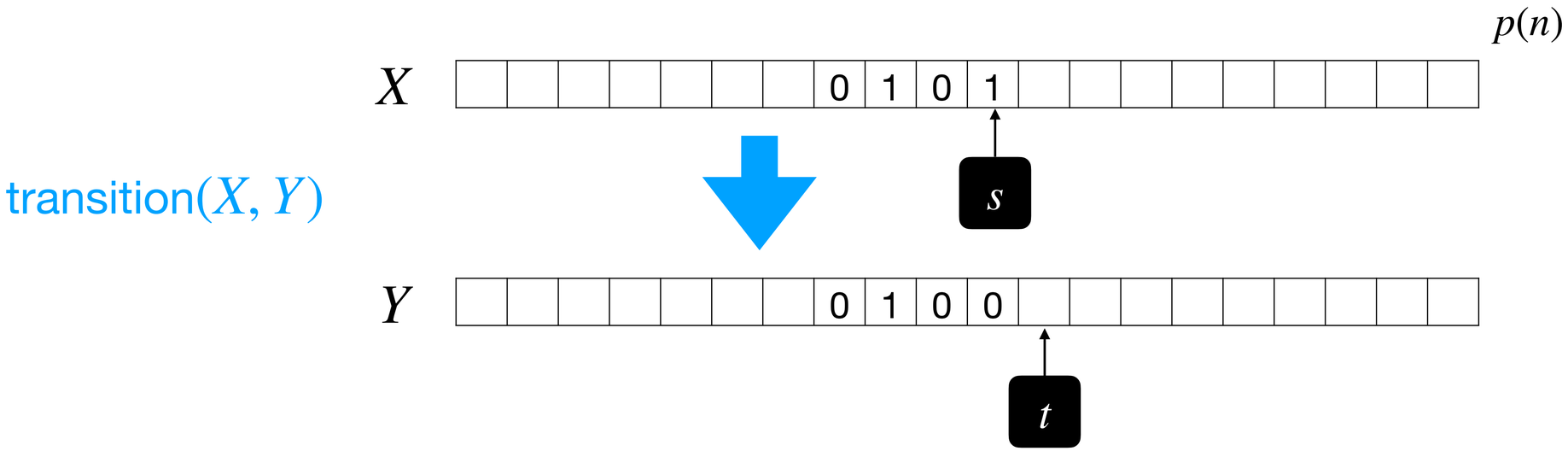
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee$

Complexity of QBF Evaluation



reach$_{\leq 1}(X, Y)$ $:=$ $X \leftrightarrow Y$ $\vee$ transition(X, Y)

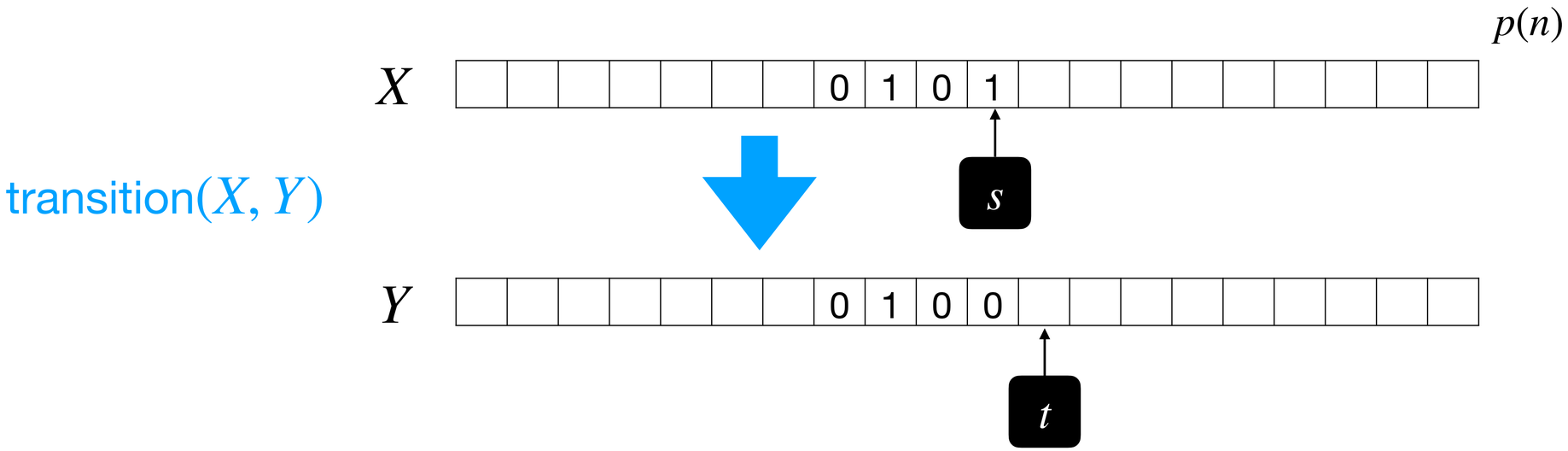
Complexity of QBF Evaluation



reach$_{\leq 1}(X, Y)$ $:=$ $X \leftrightarrow Y$ $\vee$ transition(X, Y)

reach$_{\leq 2^n}(X, Y)$

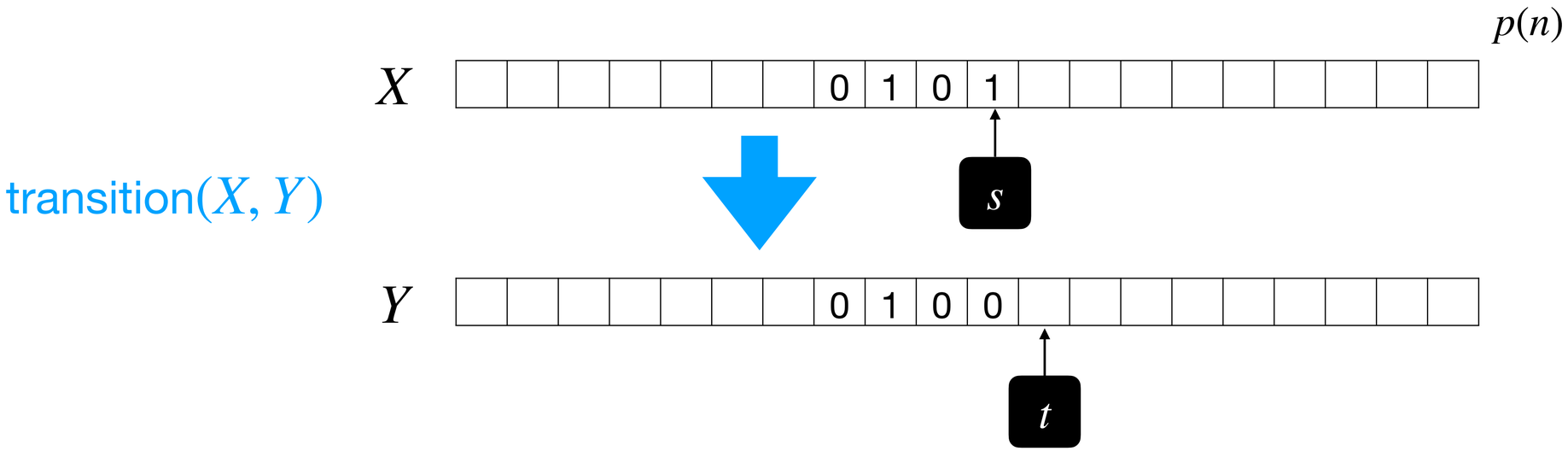
Complexity of QBF Evaluation



$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad :=$

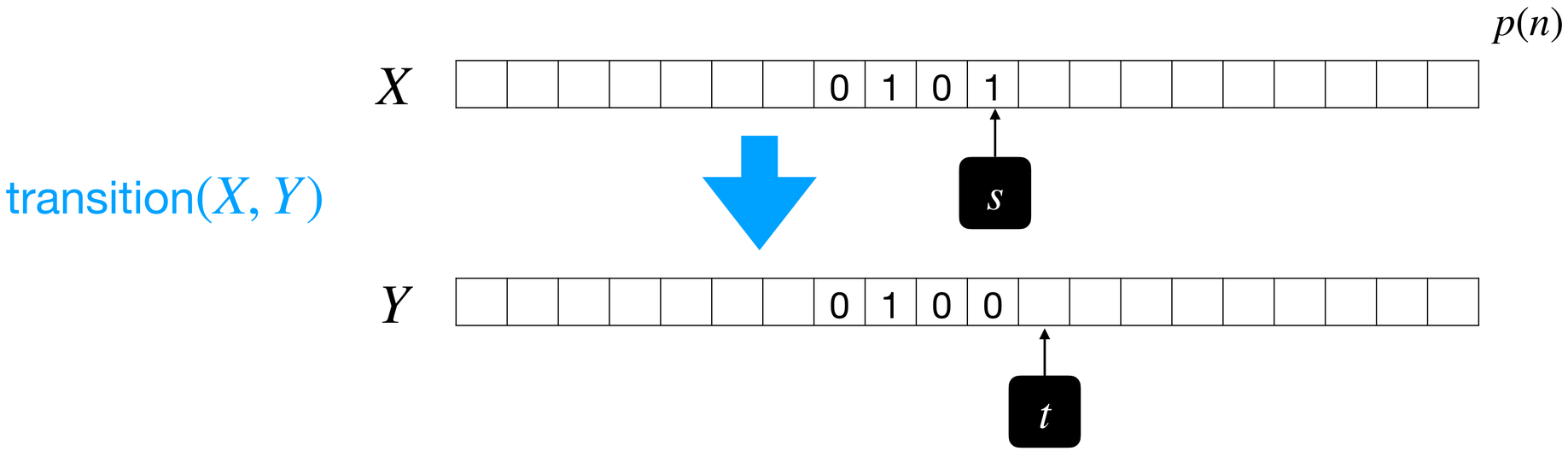
Complexity of QBF Evaluation



$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z$

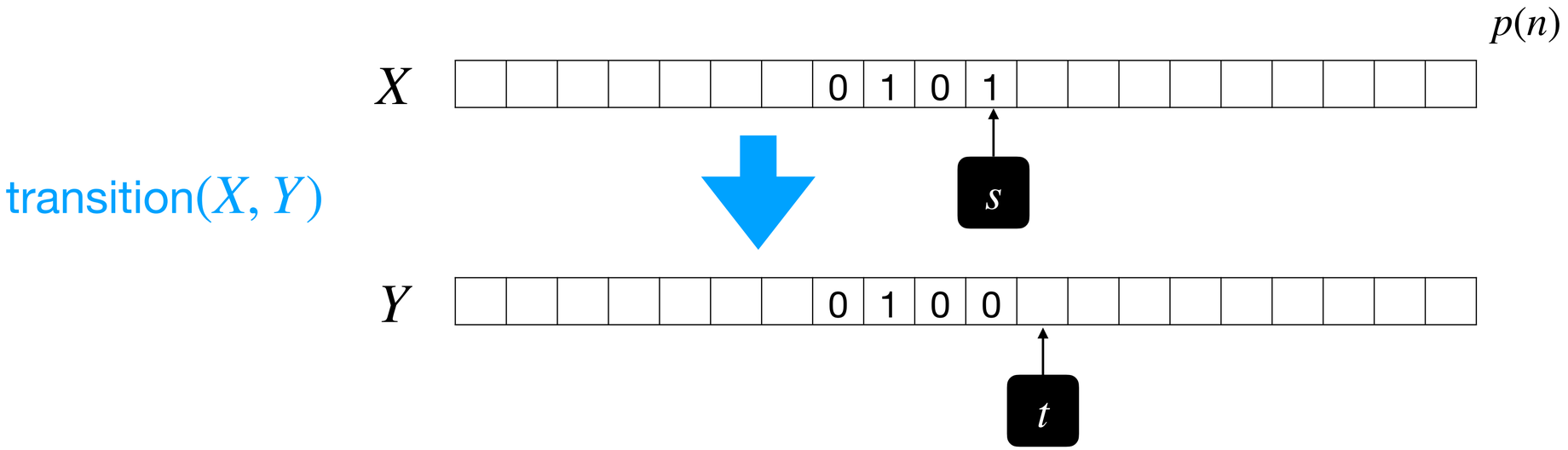
Complexity of QBF Evaluation



$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z)$

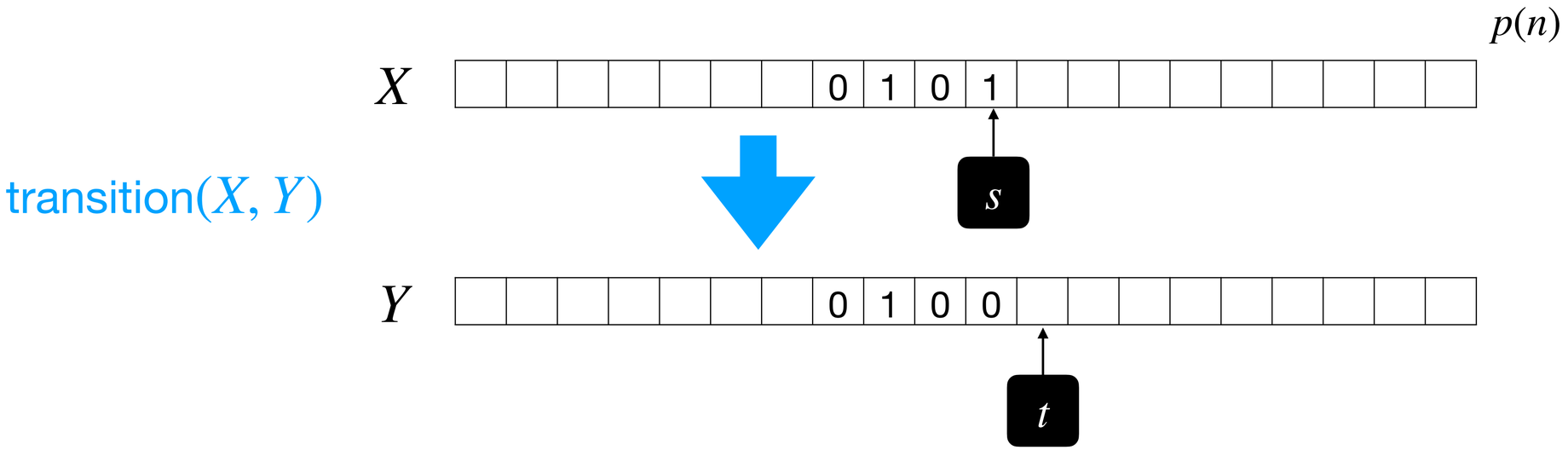
Complexity of QBF Evaluation



$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge$

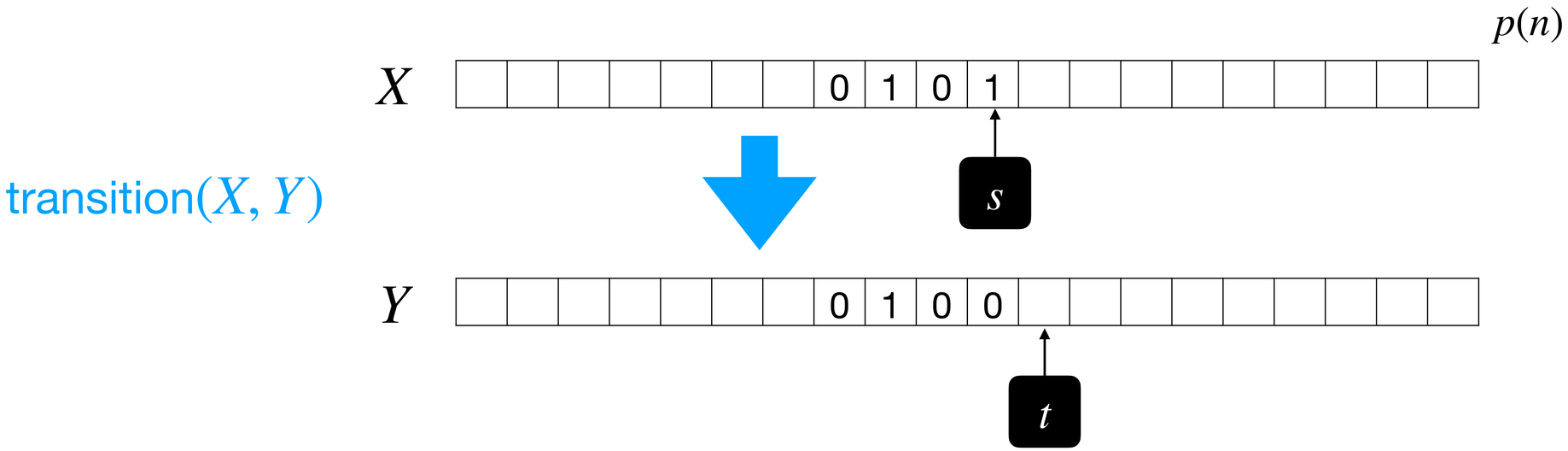
Complexity of QBF Evaluation



$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

Complexity of QBF Evaluation

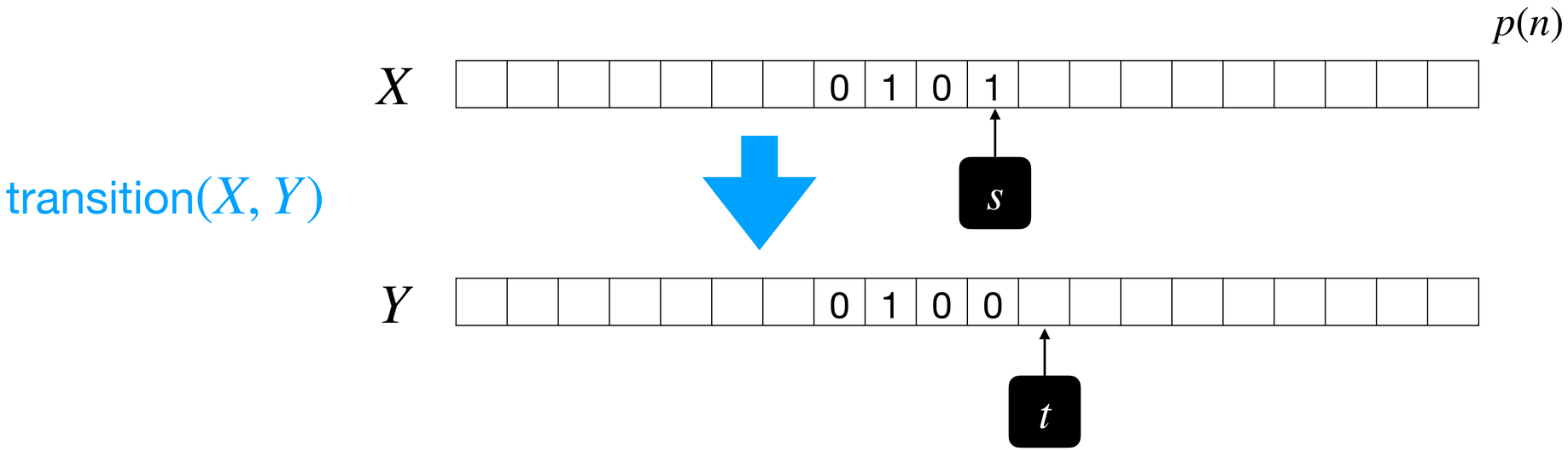


$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y)$

Complexity of QBF Evaluation

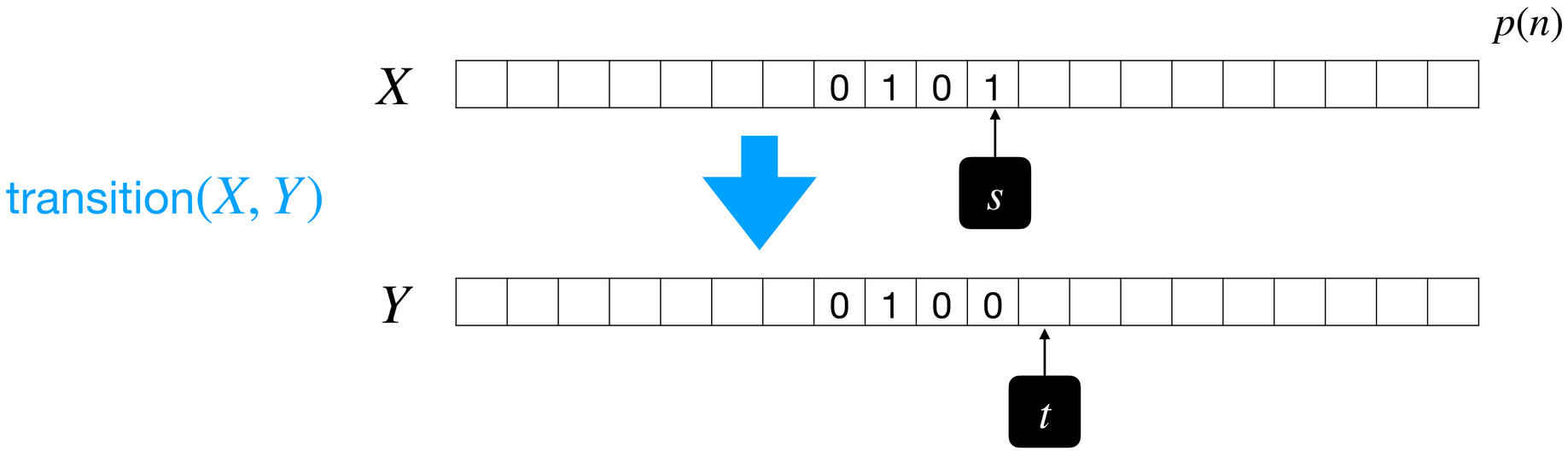


$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv$

Complexity of QBF Evaluation

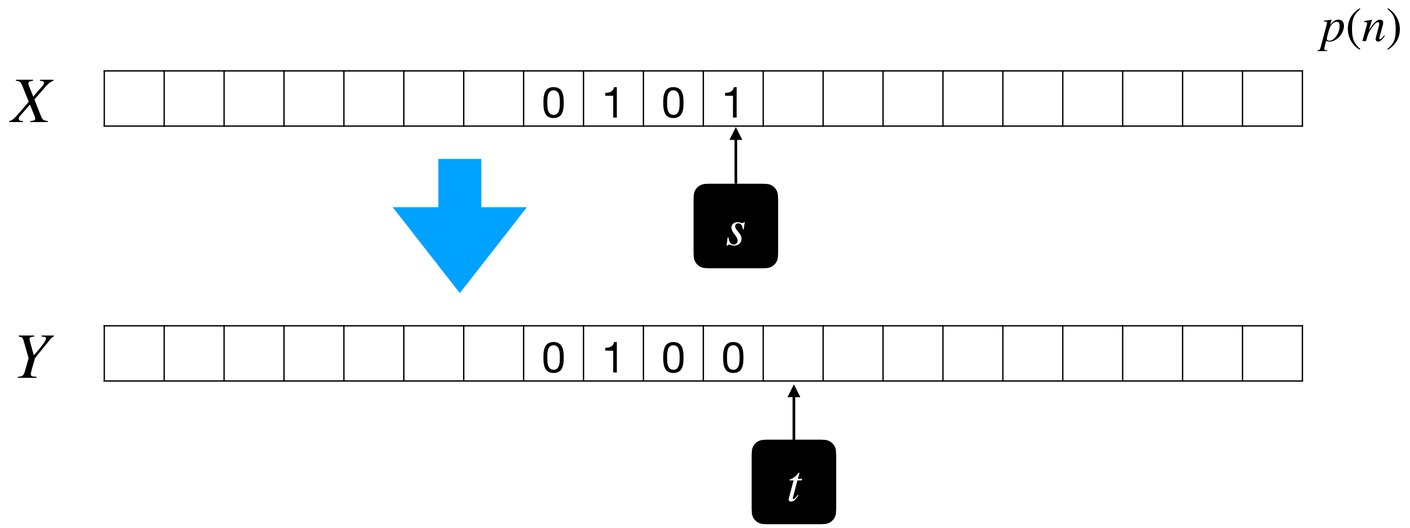


$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z$

Complexity of QBF Evaluation

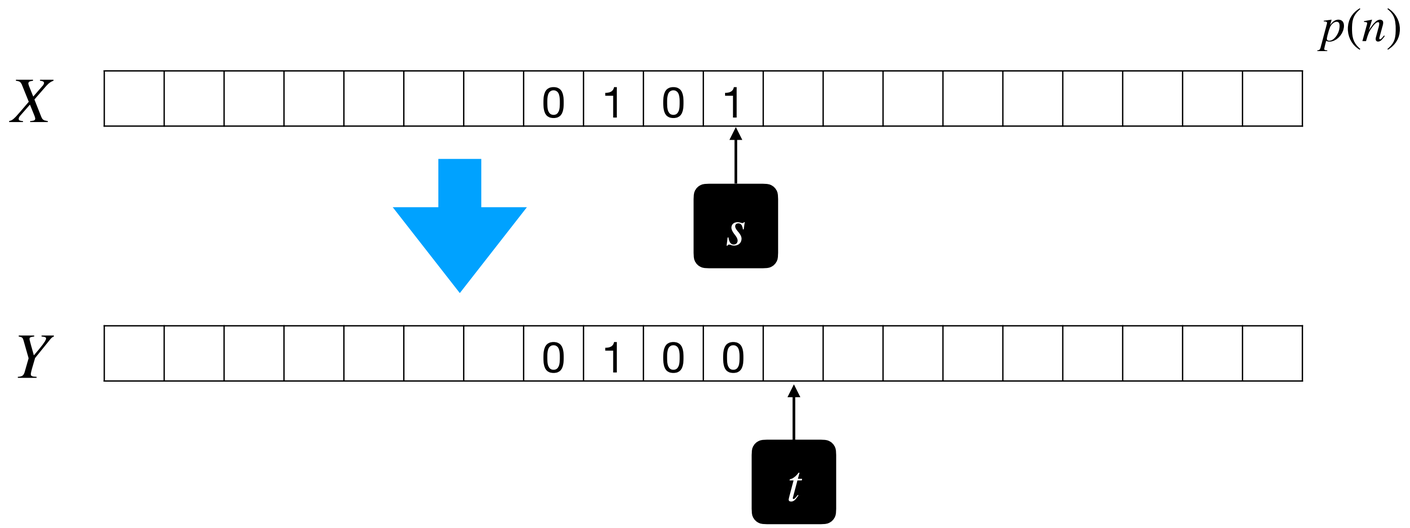


$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \quad \wedge \quad \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B$

Complexity of QBF Evaluation

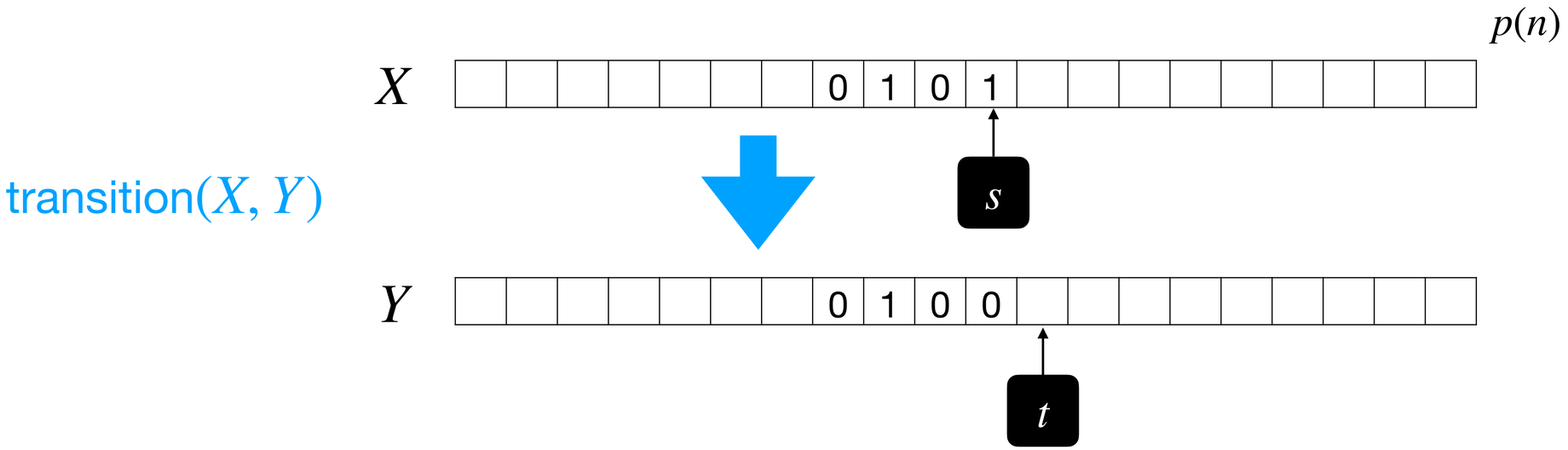


$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z)$

Complexity of QBF Evaluation

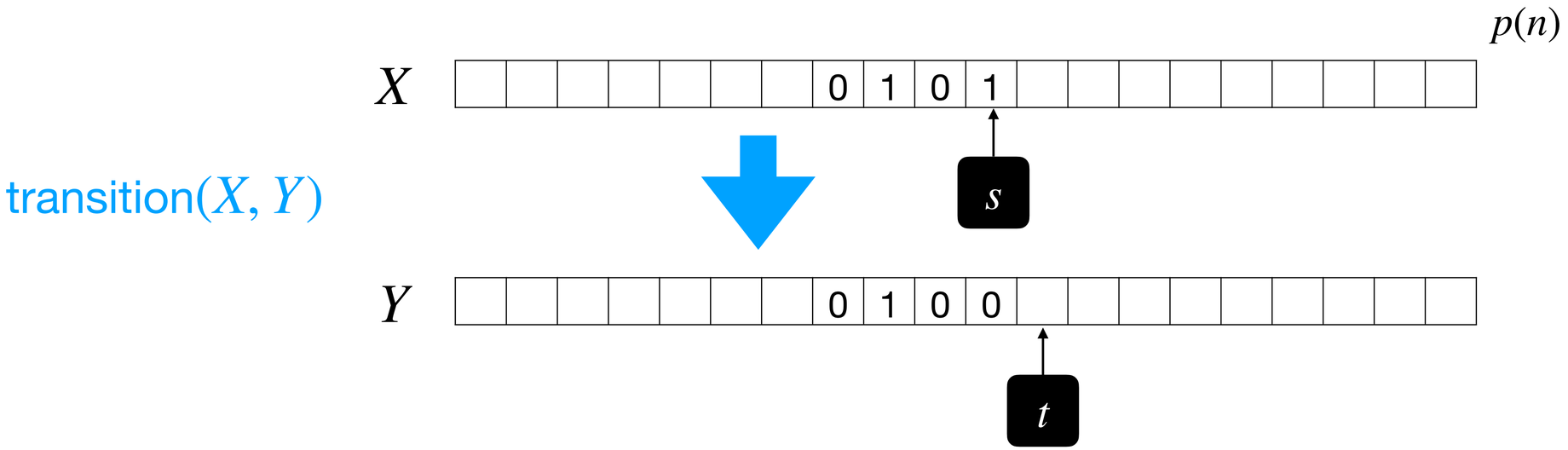


$$\text{reach}_{\leq 1}(X, Y) := X \leftrightarrow Y \lor \text{transition}(X, Y)$$

$$\text{reach}_{\leq 2n}(X, Y) := \exists Z \ \text{reach}_{\leq n}(X, Z) \land \text{reach}_{\leq n}(Z, Y)$$

$$\text{reach}_{\leq 2n}(X, Y) \equiv \exists Z \ \forall A, B \ (A \leftrightarrow X \land B \leftrightarrow Z) \lor$$

Complexity of QBF Evaluation

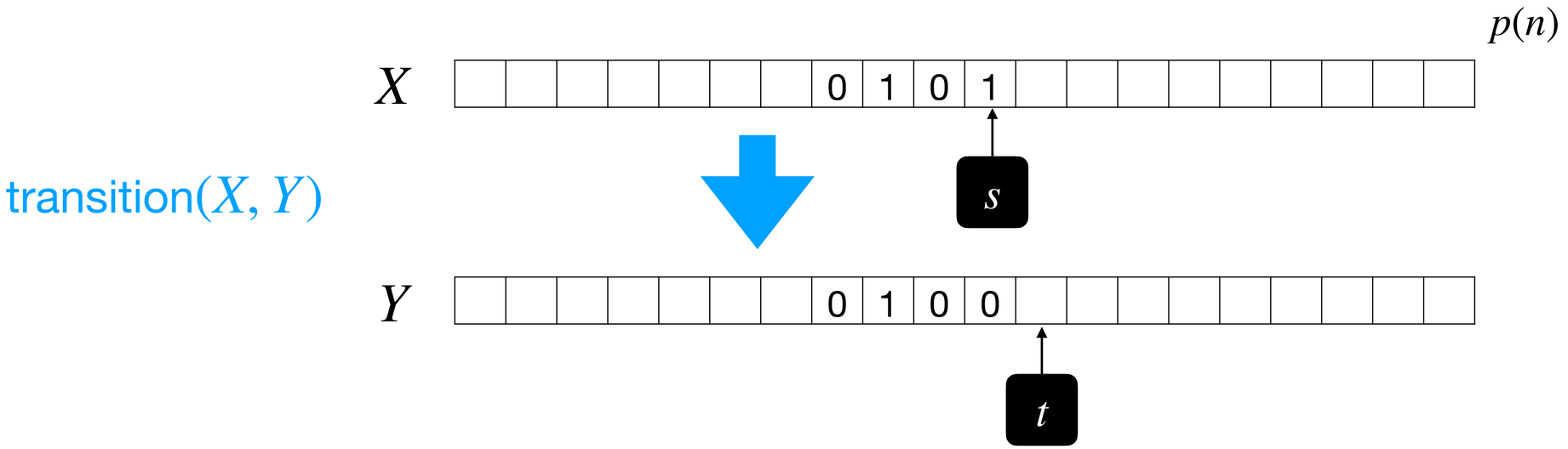


$$\text{reach}_{\leq 1}(X, Y) := X \leftrightarrow Y \lor \text{transition}(X, Y)$$

$$\text{reach}_{\leq 2n}(X, Y) := \exists Z \ \text{reach}_{\leq n}(X, Z) \land \text{reach}_{\leq n}(Z, Y)$$

$$\text{reach}_{\leq 2n}(X, Y) \equiv \exists Z \ \forall A, B \ (A \leftrightarrow X \land B \leftrightarrow Z) \lor (A \leftrightarrow Z \land B \leftrightarrow Y)$$

Complexity of QBF Evaluation

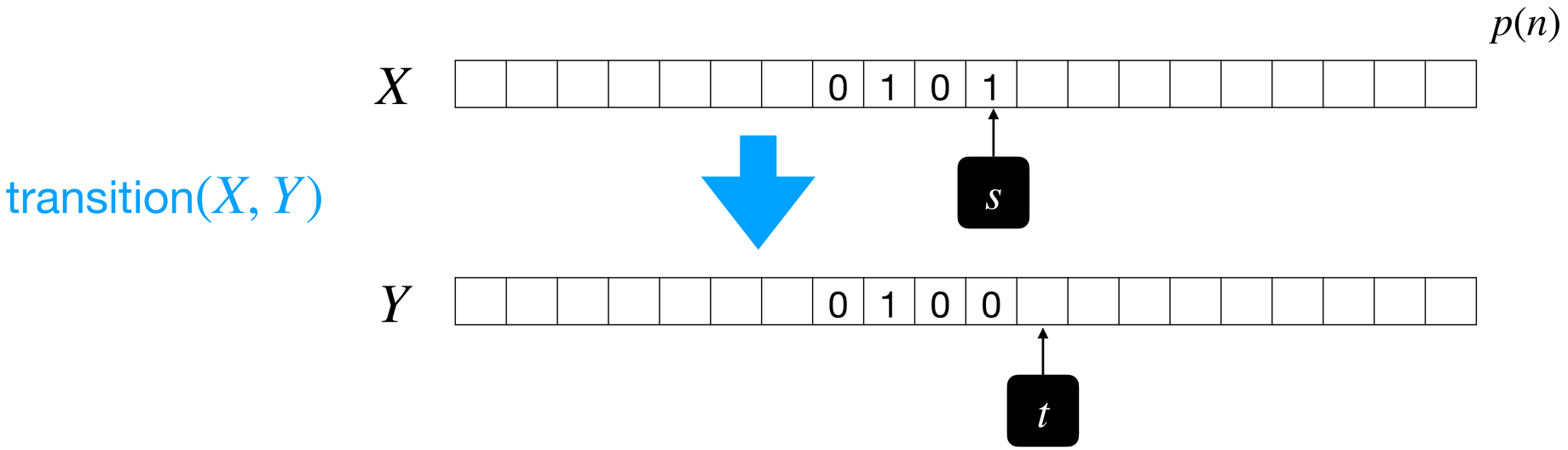


$$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$$

$$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$$

$$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow$$

Complexity of QBF Evaluation

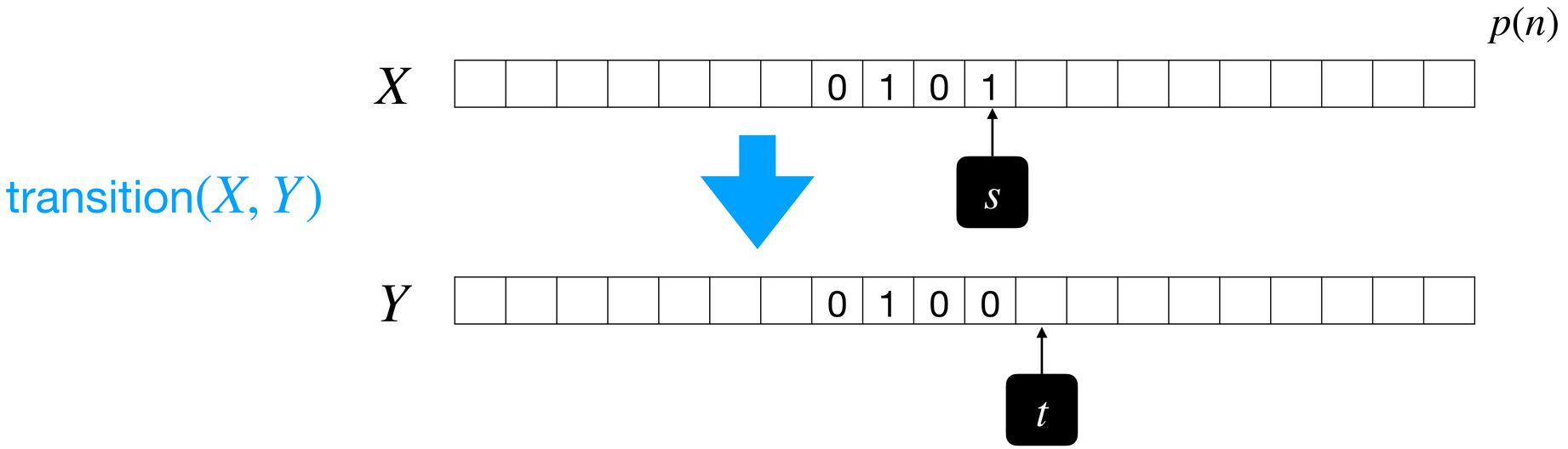


$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

Complexity of QBF Evaluation



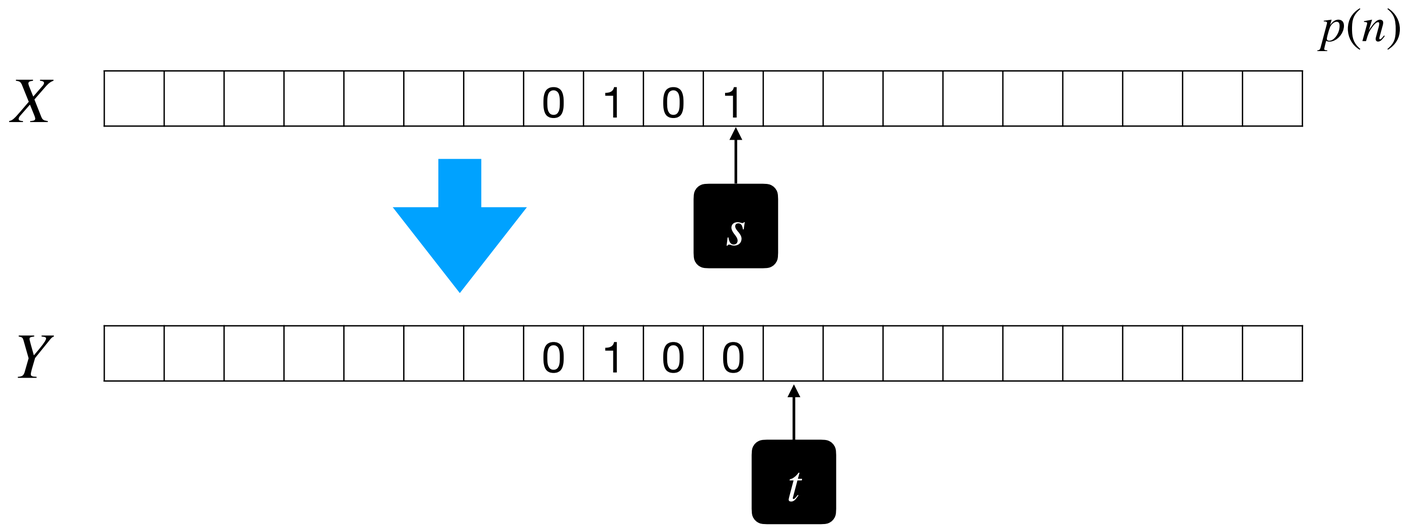
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y$

Complexity of QBF Evaluation



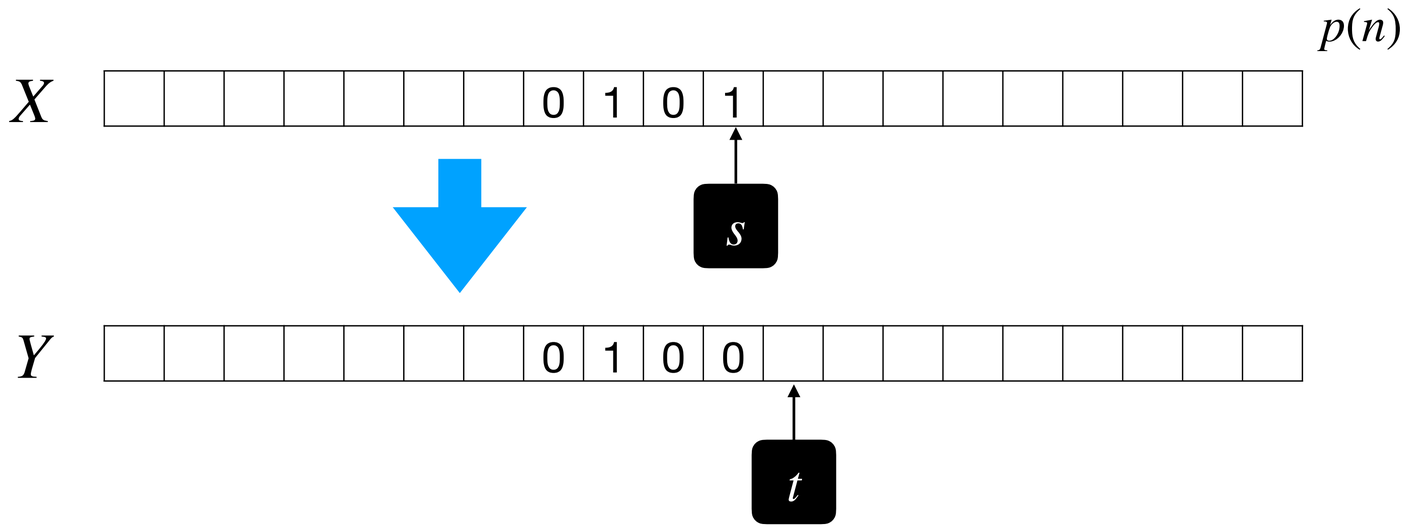
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \quad \wedge \quad \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y \;\; \text{initial}(X)$

Complexity of QBF Evaluation



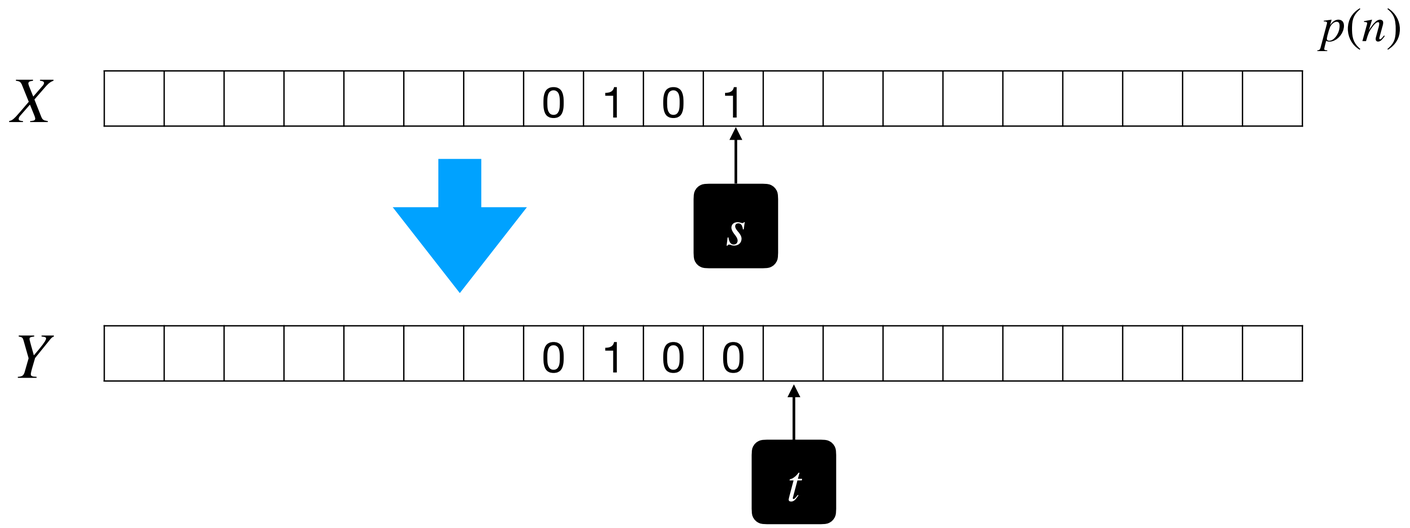
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y \;\; \text{initial}(X) \;\; \wedge$

Complexity of QBF Evaluation



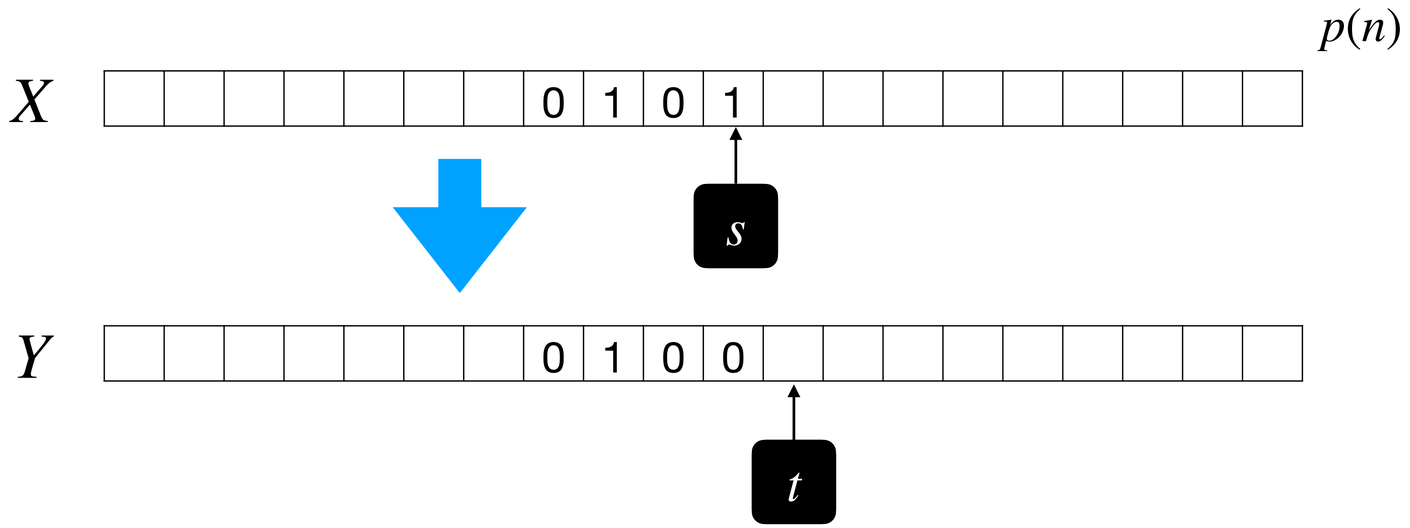
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \quad \wedge \quad \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y \;\; \text{initial}(X) \;\; \wedge \;\; \text{accepting}(Y)$

Complexity of QBF Evaluation



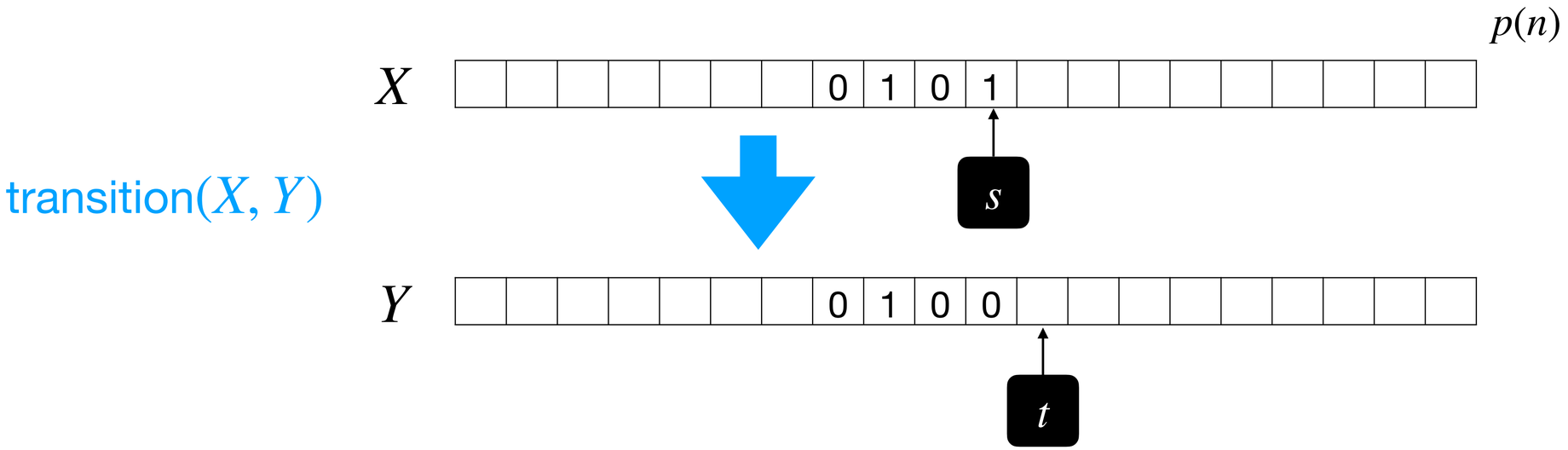
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \quad \wedge \quad \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y \;\; \text{initial}(X) \;\; \wedge \;\; \text{accepting}(Y) \;\; \wedge$

Complexity of QBF Evaluation



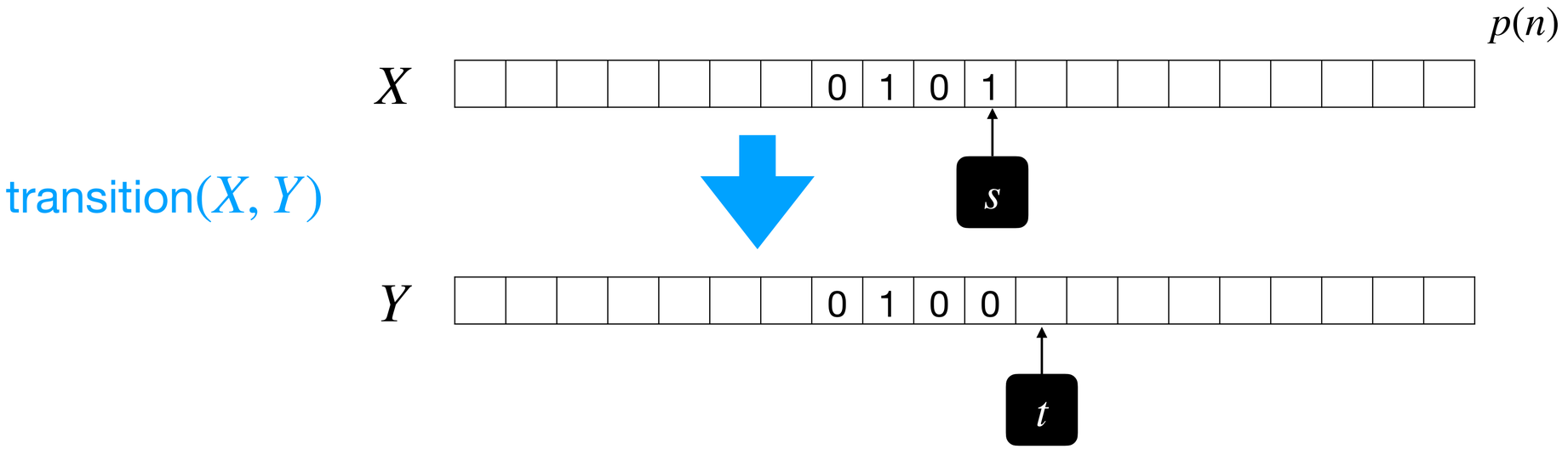
$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \;\; \wedge \;\; \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y \;\; \text{initial}(X) \;\; \wedge \;\; \text{accepting}(Y) \;\; \wedge \;\; \text{reach}_{\leq 2^{|X|}}(X, Y)$

Complexity of QBF Evaluation



$\text{reach}_{\leq 1}(X, Y) \quad := \quad X \leftrightarrow Y \quad \vee \quad \text{transition}(X, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad := \quad \exists Z \;\; \text{reach}_{\leq n}(X, Z) \quad \wedge \quad \text{reach}_{\leq n}(Z, Y)$

$\text{reach}_{\leq 2n}(X, Y) \quad \equiv \quad \exists Z \;\; \forall A, B \;\; (A \leftrightarrow X \wedge B \leftrightarrow Z) \;\; \vee \;\; (A \leftrightarrow Z \wedge B \leftrightarrow Y) \;\; \rightarrow \;\; \text{reach}_{\leq n}(A, B)$

$\exists X \exists Y \;\; \text{initial}(X) \;\; \wedge \;\; \text{accepting}(Y) \;\; \wedge \;\; \text{reach}_{\leq 2^{|X|}}(X, Y)$ QSAT is **PSPACE**-hard

Algorithms

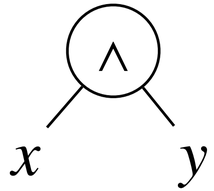
Prenex Conjunctive Normal Form

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

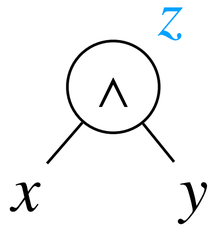
Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



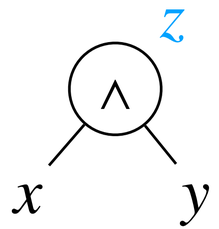
Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$



Prenex Conjunctive Normal Form

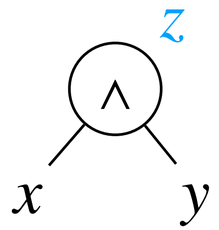
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$(x \vee \neg z)$

Prenex Conjunctive Normal Form

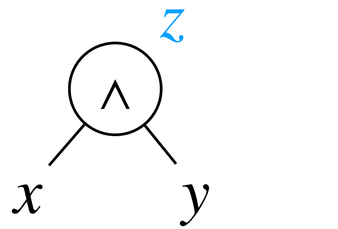
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$(x \vee \neg z) \wedge$

Prenex Conjunctive Normal Form

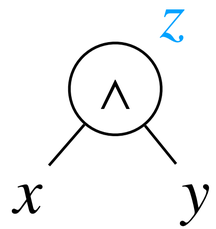
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z)$$

Prenex Conjunctive Normal Form

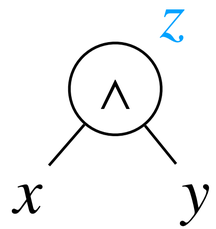
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

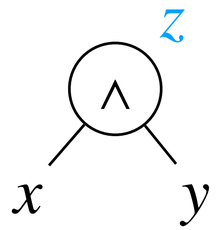


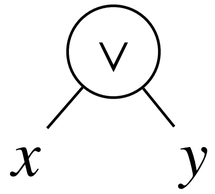
$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge$$

$$(\neg x \vee \neg y \vee z)$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



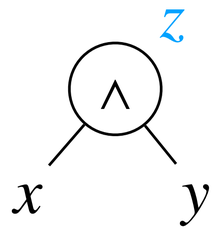


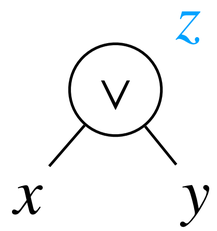
$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge$$

$$(\neg x \vee \neg y \vee z)$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



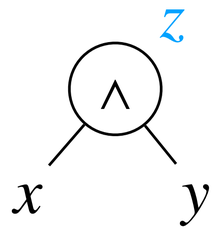


$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge$$

$$(\neg x \vee \neg y \vee z)$$

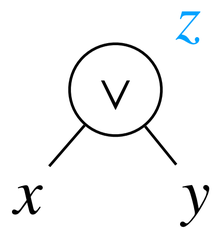
Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge$$

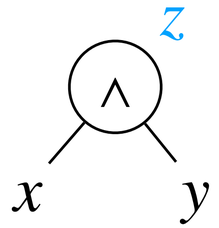
$$(\neg x \vee \neg y \vee z)$$



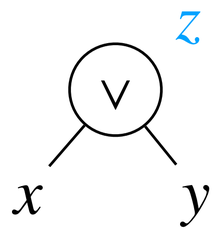
$$(\neg x \vee z)$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



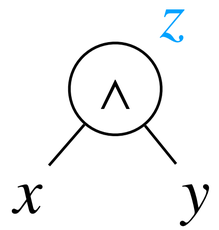
$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge$$
$$(\neg x \vee \neg y \vee z)$$



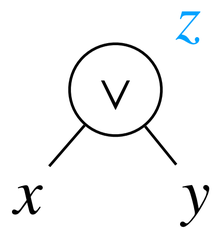
$$(\neg x \vee z) \wedge$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



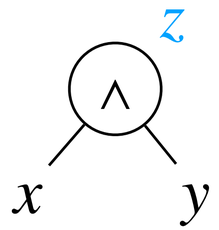
$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



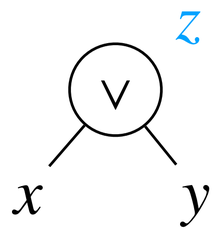
$$(\neg x \vee z) \wedge (\neg y \vee z)$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



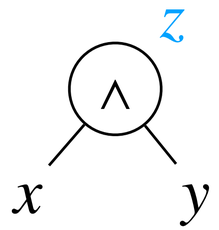
$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



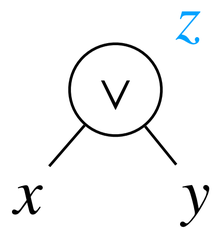
$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge$$

Prenex Conjunctive Normal Form

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



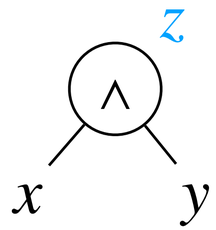
$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



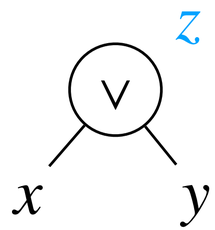
$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$

Prenex Conjunctive Normal Form

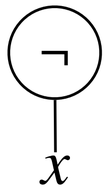
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$

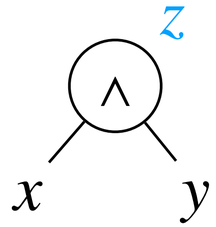


$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$

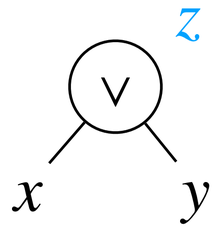


Prenex Conjunctive Normal Form

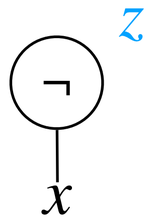
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$

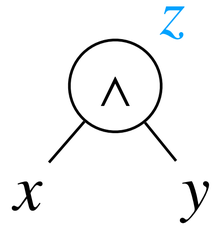


$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$

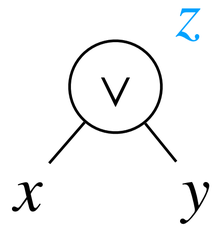


Prenex Conjunctive Normal Form

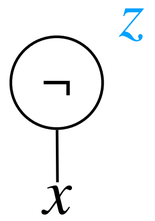
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



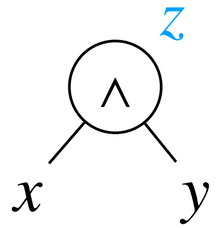
$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$



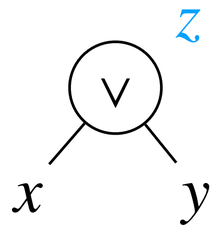
$$(x \vee z)$$

Prenex Conjunctive Normal Form

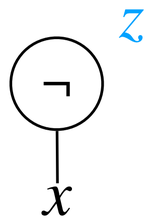
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



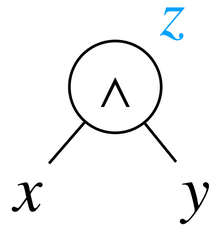
$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$



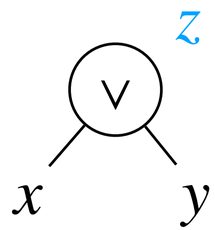
$$(x \vee z) \wedge$$

Prenex Conjunctive Normal Form

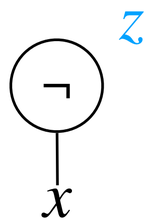
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



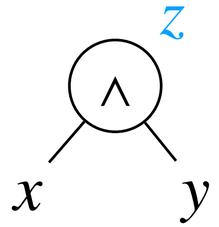
$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$



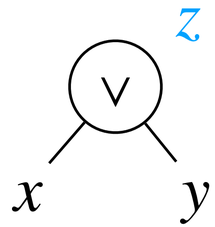
$$(x \vee z) \wedge (\neg x \vee \neg z)$$

Prenex Conjunctive Normal Form

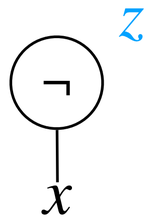
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$

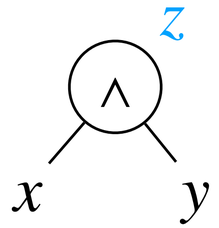


$$(x \vee z) \wedge (\neg x \vee \neg z)$$

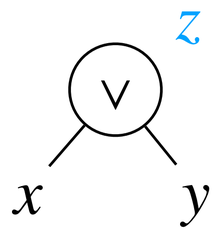
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n$$

Prenex Conjunctive Normal Form

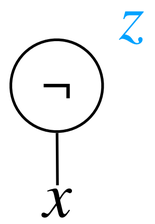
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$

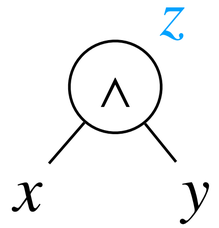


$$(x \vee z) \wedge (\neg x \vee \neg z)$$

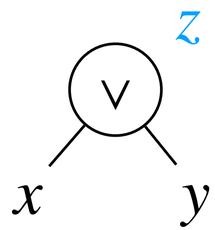
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \; \exists z_1 \ldots \exists z_m$$

Prenex Conjunctive Normal Form

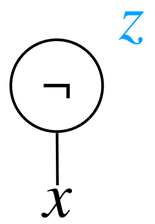
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$

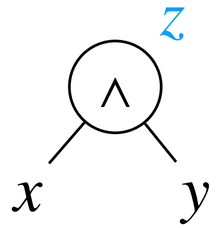


$$(x \vee z) \wedge (\neg x \vee \neg z)$$

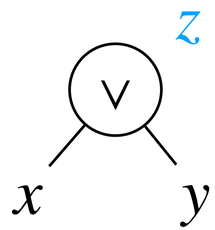
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \; \exists z_1 \ldots \exists z_m \; \psi$$

Prenex Conjunctive Normal Form

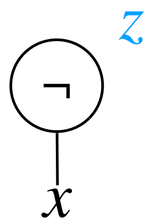
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$



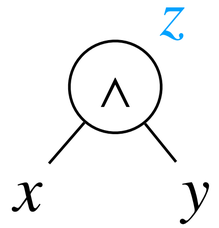
$$(x \vee z) \wedge (\neg x \vee \neg z)$$

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \; \exists z_1 \ldots \exists z_m \; \psi$$

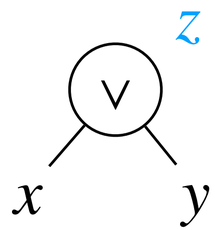
CNF

Prenex Conjunctive Normal Form

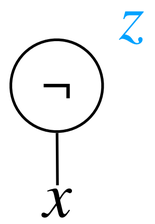
$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$



$$(x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$$



$$(\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$$



$$(x \vee z) \wedge (\neg x \vee \neg z)$$

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \; \exists z_1 \ldots \exists z_m \; \psi$$

CNF

Conversion to PCNF can make solving harder!

1. Quantified DPLL

Quantified DPLL for PCNF

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

If $\mathbf{Q}$ only contains existential quantifiers, return **true** if and only if φ is satisfiable.

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

If $\mathbf{Q}$ only contains existential quantifiers, return **true** if and only if φ is satisfiable.

Delete all universally quantified literals. If the resulting propositional formula is satisfiable, return **true**.

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

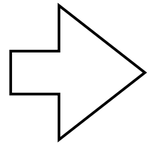
Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ

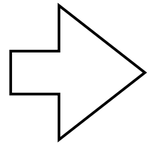
Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ 

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ  set x to $\top$ ($\bot$)

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ ⇨ set x to $\top$ ($\bot$)

$\forall x$ only occurs positively (negatively) in φ

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ ⇨ set x to $\top$ ($\bot$)

$\forall x$ only occurs positively (negatively) in φ ⇨

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ ⇨ set x to $\top$ ($\bot$)

$\forall x$ only occurs positively (negatively) in φ ⇨ set x to $\bot$ ($\top$)

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

$\exists x$ only occurs positively (negatively) in φ ⇨ set x to $\top$ ($\bot$)

$\forall x$ only occurs positively (negatively) in φ ⇨ set x to $\bot$ ($\top$)

“Pure Literals”

Quantified DPLL for PCNF

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
```

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

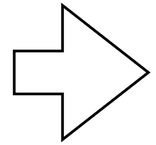
Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

clause C only contains universally quantified variables

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

clause C only contains universally quantified variables 

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

clause C only contains universally quantified variables ⇨ return **False**

Universal Reduction

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$C \vee x_i$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$C \vee x_i$ $Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$C \vee x_i$

$\downarrow$

C

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$C \vee x_i$ (with $\downarrow C$)

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

$\exists$-winning strategy $\mathbf{f}$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$$C \vee x_i \quad \downarrow \quad C$$

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

$\exists$-winning strategy $\mathbf{f}$

$\forall$-play τ

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$$C \vee x_i \quad \downarrow \quad C$$

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

$\exists$-winning strategy $\mathbf{f}$

$\forall$-play τ

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$$C \vee x_i \quad \Big\downarrow \quad C$$

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

assume π **falsifies** C

$\exists$-winning strategy $\mathbf{f}$

$\forall$-play τ

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$$C \vee x_i \quad \downarrow \quad C$$

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

assume π **falsifies** C

$\exists$-winning strategy $\mathbf{f}$

$\forall$-play τ

$\forall$-play τ' with $\tau'(x_i) = \bot$

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$C \vee x_i$ (with $\downarrow C$)

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

assume π **falsifies** C

$\exists$-winning strategy $\mathbf{f}$

$\forall$-play τ

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

$\forall$-play τ' with $\tau'(x_i) = \bot$

combined play $\pi' = \mathbf{f}(\tau') \cup \tau'$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \quad \varphi$$

$C \vee x_i$ (with $\downarrow C$)

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

assume π **falsifies** C

∃-winning strategy $\mathbf{f}$

∀-play τ

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

∀-play τ' with $\tau'(x_i) = \bot$

combined play $\pi' = \mathbf{f}(\tau') \cup \tau'$

$\mathbf{f}(\tau)$ and $\mathbf{f}(\tau')$ must agree on $\{x_1, \ldots, x_{i-1}\}$

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$C \vee x_i$ $\quad Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

$\downarrow$

C

assume π **falsifies** C

$\exists$-winning strategy $\mathbf{f}$

$\forall$-play τ

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

$\forall$-play τ' with $\tau'(x_i) = \bot$

combined play $\pi' = \mathbf{f}(\tau') \cup \tau'$

$\mathbf{f}(\tau)$ and $\mathbf{f}(\tau')$ must agree on $\{x_1, \ldots, x_{i-1}\}$

$C \vee x_i$ falsified by π'

Universal Reduction

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; \varphi$$

$C \vee x_i$ ↓ C

$Q_i = \forall \wedge var(C) \subseteq \{x_1, \ldots, x_{i-1}\}$

assume π **falsifies** C

∃-winning strategy $\mathbf{f}$

∀-play τ

combined play $\pi = \mathbf{f}(\tau) \cup \tau$

∀-play τ' with $\tau'(x_i) = \bot$

combined play $\pi' = \mathbf{f}(\tau') \cup \tau'$

$\mathbf{f}(\tau)$ and $\mathbf{f}(\tau')$ must agree on $\{x_1, \ldots, x_{i-1}\}$

$C \vee x_i$ falsified by π' contradiction!

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
    universal_reduction()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

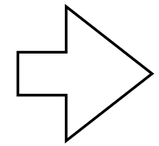
Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
    universal_reduction()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

unit clause (l)

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
    universal_reduction()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

unit clause (l) 

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
    universal_reduction()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

unit clause (l) ⇨ $l \leftarrow \top$

Quantified DPLL for PCNF

```
def evaluate(Q, φ):
  apply_SAT_rules()
  Q_old = None
  while Q != [] and Q != Q_old:
    Q_old = Q
    assign_pure_literals()
    universal_reduction()
    propagate_units()
  if Q == []:
    return φ == []
  elif Q_1 == ∃:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) or evaluate(Q[1:], φ[x_1 ← ⊥])
  else:
    return evaluate(Q[1:], φ[x_1 ← ⊤]) and evaluate(Q[1:], φ[x_1 ← ⊥])
```

2. Quantified CDCL

From CDCL to QCDCL

From CDCL to QCDCL

```
def CDCL():
```

From CDCL to QCDCL

```
def CDCL():
  while True:
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

From CDCL to QCDCL

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$$x \stackrel{d}{=} \top$$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$$x \stackrel{d}{=} \top$$

$$(\neg x \vee y)$$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$$x \stackrel{d}{=} \top \qquad\qquad y = \top$$

$$(\neg x \vee y)$$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$$x \overset{d}{=} \top \qquad\qquad y = \top$$

$$(\neg x \vee y) \qquad (\neg x \vee \neg y)$$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$$x \overset{d}{=} \top \qquad y = \top \qquad \varnothing$$

$$(\neg x \vee y) \qquad (\neg x \vee \neg y)$$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$$x \overset{d}{=} \top \qquad y = \top \qquad \varnothing$$

$$(\neg x \vee y) \qquad (\neg x \vee \neg y)$$

$$(\neg x)$$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$x \overset{d}{=} \top$ $\quad$ $y = \top$ $\quad$ $\varnothing$

$(\neg x \vee y)$ $\quad$ $(\neg x \vee \neg y)$

$(\neg x)$

$(\neg x)$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$x \overset{d}{=} \top$ $\quad$ $y = \top$ $\quad$ $\varnothing$

$(\neg x \vee y)$ $\quad$ $(\neg x \vee \neg y)$

$(\neg x)$

$x = \bot$

$(\neg x)$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$x \overset{d}{=} \top$ $\quad$ $y = \top$ $\quad$ $\varnothing$

$(\neg x \vee y)$ $\quad$ $(\neg x \vee \neg y)$

$(\neg x)$

$x = \bot$

$(\neg x)$ $\quad$ $(x \vee y)$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

$x \overset{d}{=} \top$ $\quad$ $y = \top$ $\quad$ $\varnothing$

$(\neg x \vee y)$ $\quad$ $(\neg x \vee \neg y)$

$(\neg x)$

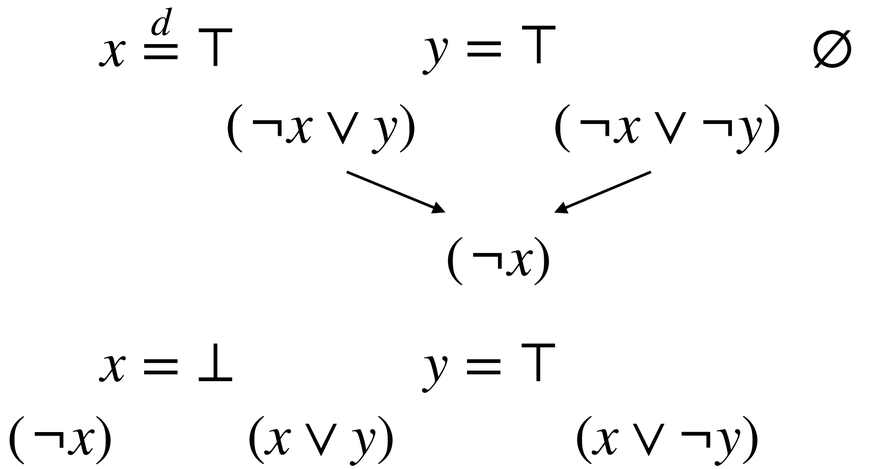
$x = \bot$ $\quad$ $y = \top$

$(\neg x)$ $\quad$ $(x \vee y)$

From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

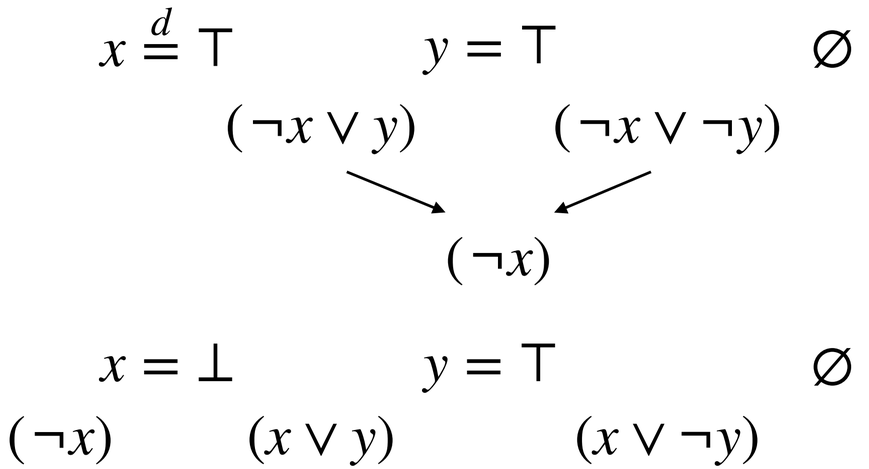
$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$



From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

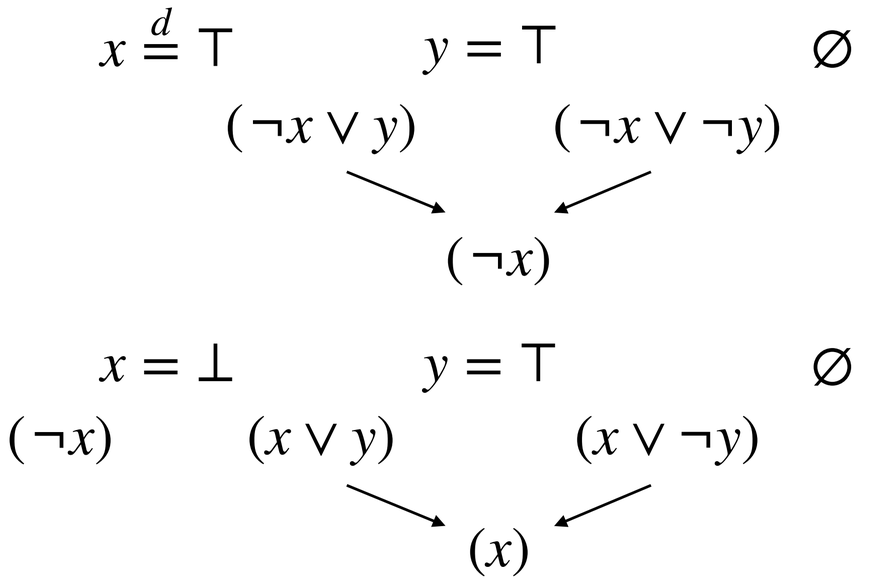
$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$



From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

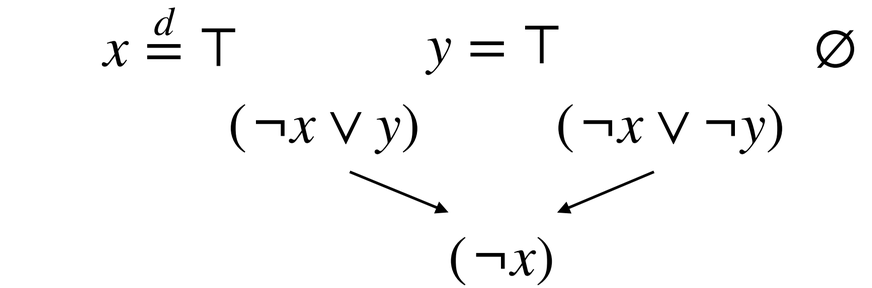
$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$

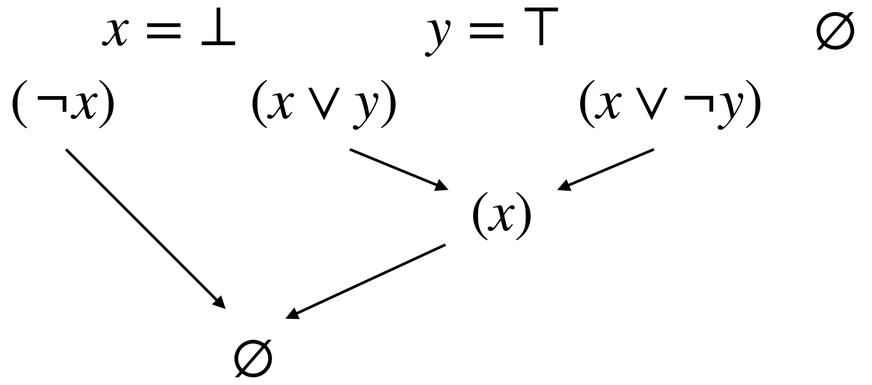


From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$



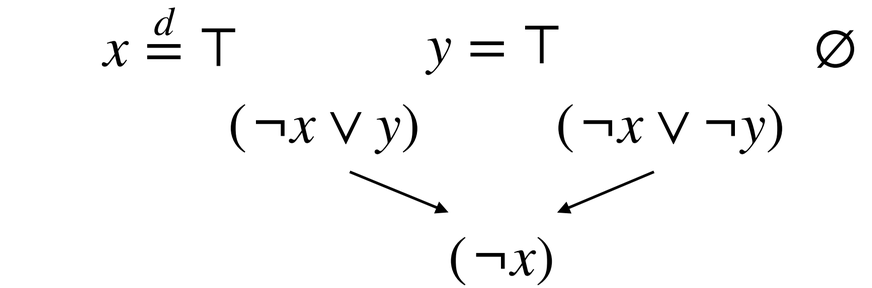


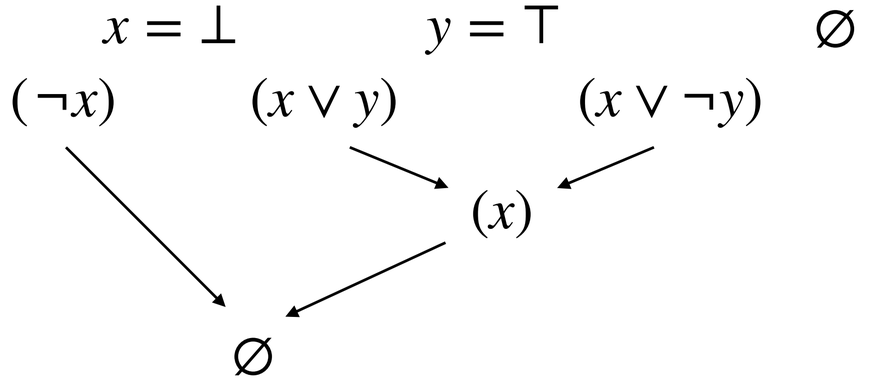
From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

1

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$





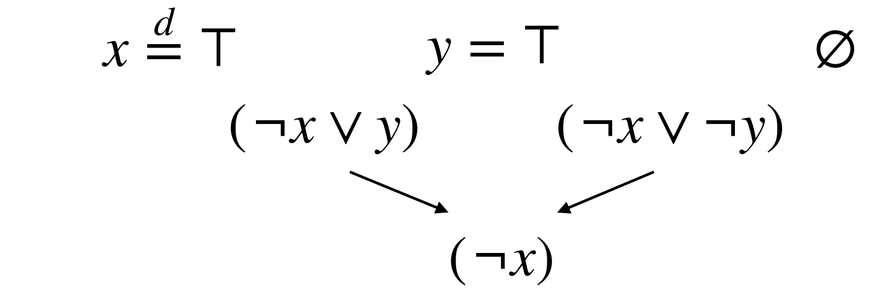
From CDCL to QCDCL

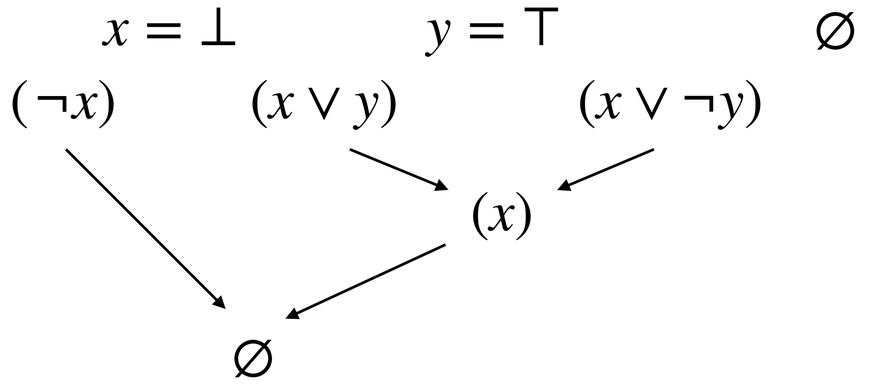
```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

1

2

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$



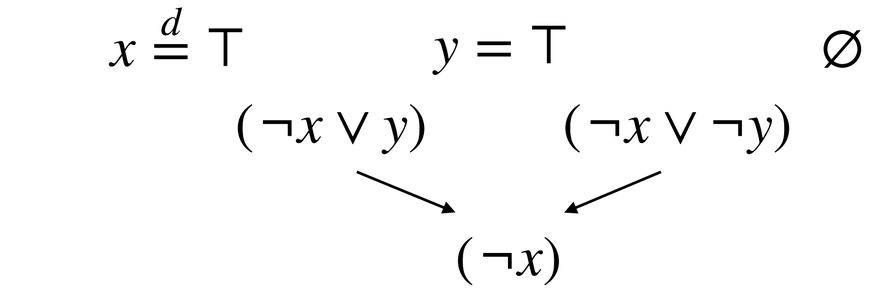


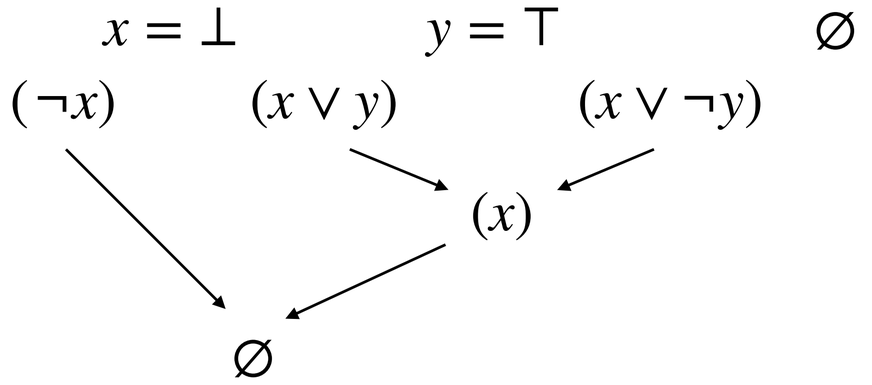
From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

1 2 3

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$



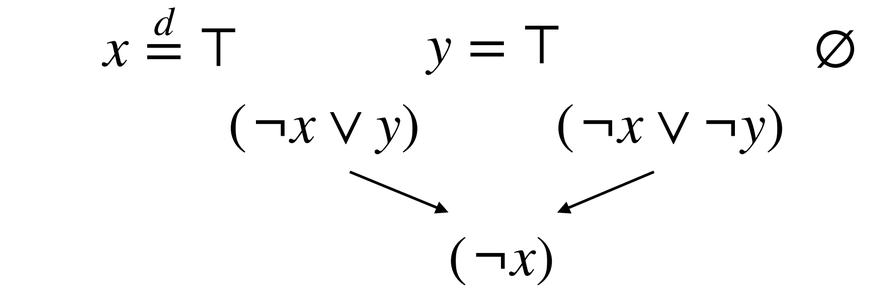


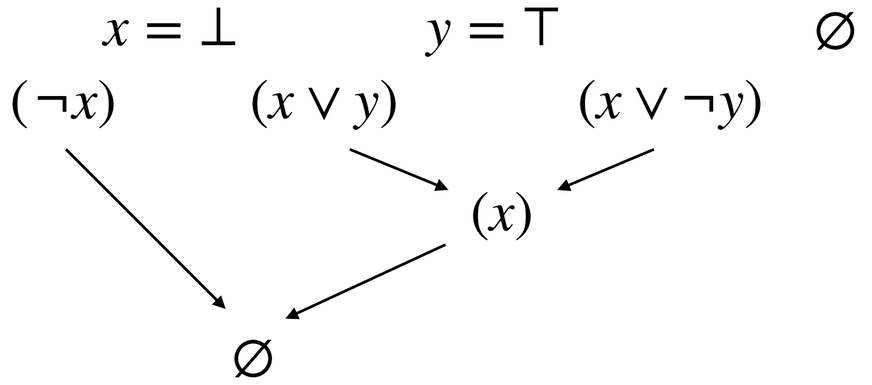
From CDCL to QCDCL

```
def CDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if clause == []:
        return False
      attach(clause)
      backtrack(bt_level)
    elif allAssigned():
      return True
    else:
      decide()
```

1 2 3 4

$$(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$$





QCDCL with Propositional Unit Propagation

QCDCL with Propositional Unit Propagation

```
def QCDCL():
```

QCDCL with Propositional Unit Propagation

```
def QCDCL():
  while True:
```

QCDCL with Propositional Unit Propagation

```
def QCDCL():
  while True:
    conflict = propagate()
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \overset{d}{=} \top$$

QCDCL with Propositional Unit Propagation

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Propagate existential units.
Conflict at purely universal clauses.

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \overset{d}{=} \top$$

$$(\neg e_1 \vee \neg e_4)$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \stackrel{d}{=} \top \qquad e_4 = \bot$$

$$(\neg e_1 \vee \neg e_4)$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \stackrel{d}{=} \top \qquad e_4 = \bot$$

$$(\neg e_1 \vee \neg e_4) \qquad (u \vee e_4)$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \stackrel{d}{=} \top \qquad e_4 = \bot$$

$$(\neg e_1 \vee \neg e_4) \qquad (u \vee e_4)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2}$$

QCDCL with Propositional Unit Propagation

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Propagate existential units.
Conflict at purely universal clauses.

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \stackrel{d}{=} \top \qquad e_4 = \bot$$

$$(\neg e_1 \vee \neg e_4) \qquad (u \vee e_4)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \overset{d}{=} \top \qquad e_4 = \bot$$

$$(\neg e_1 \vee \neg e_4) \qquad (u \vee e_4)$$

$$(\neg e_1 \vee u)$$

Q-Resolution

$$C_1 \vee x \qquad \neg x \vee C_2 \qquad C \vee u$$

$$C_1 \vee C_2 \qquad C$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 \overset{d}{=} \top \qquad e_4 = \bot$$

$$\frac{(\neg e_1 \vee \neg e_4) \qquad (u \vee e_4)}{(\neg e_1 \vee u)}$$

$$\downarrow$$

$$(\neg e_1)$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg e_1)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 = \bot$$

$$(\neg e_1)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 = \bot$$

$$(\neg e_1) \qquad (e_1 \vee \neg e_3)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 = \bot \qquad e_3 = \bot$$

$$(\neg e_1) \qquad (e_1 \vee \neg e_3)$$

Q-Resolution

$$C_1 \vee x \qquad \neg x \vee C_2 \qquad C \vee u$$

$$\searrow \quad \swarrow \qquad \downarrow$$

$$C_1 \vee C_2 \qquad C$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 = \bot \qquad e_3 = \bot$$

$$(\neg e_1) \qquad (e_1 \vee \neg e_3) \qquad (e_2 \vee e_3)$$

Q-Resolution

$$C_1 \vee x \qquad \neg x \vee C_2 \qquad C \vee u$$

$$\searrow \quad \swarrow \qquad \downarrow$$

$$C_1 \vee C_2 \qquad C$$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

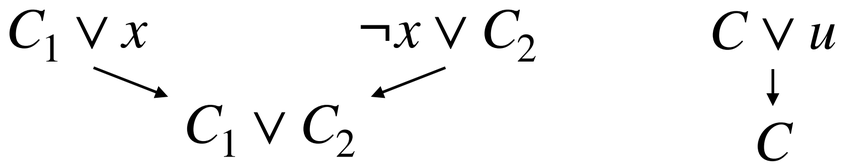
$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_1 = \bot \qquad e_3 = \bot \qquad e_2 = \top$$

$$(\neg e_1) \qquad (e_1 \vee \neg e_3) \qquad (e_2 \vee e_3)$$

Q-Resolution



QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

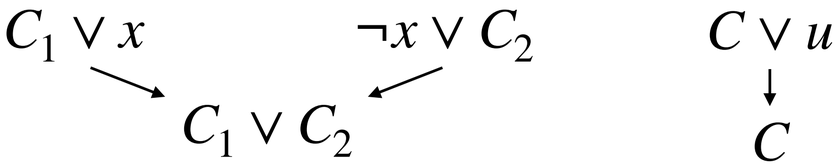
$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$e_1 = \bot$ $e_3 = \bot$ $e_2 = \top$

$(\neg e_1)$ $(e_1 \vee \neg e_3)$ $(e_2 \vee e_3)$ $(\neg e_2 \vee \neg u \vee e_3)$

Q-Resolution



QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$e_1 = \bot$ $(\neg e_1)$

$e_3 = \bot$ $(e_1 \vee \neg e_3)$

$e_2 = \top$ $(e_2 \vee e_3)$ $(\neg e_2 \vee \neg u \vee e_3)$

$(\neg u \vee e_3)$

Q-Resolution

$C_1 \vee x$ $\quad \neg x \vee C_2$ $\quad \Rightarrow \quad C_1 \vee C_2$

$C \vee u \quad \Rightarrow \quad C$

QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

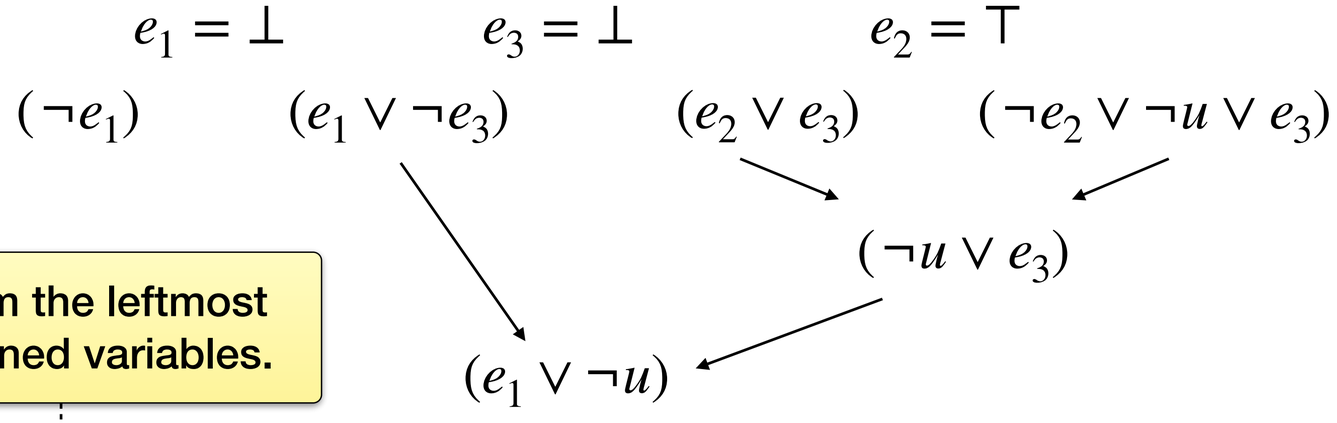
```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

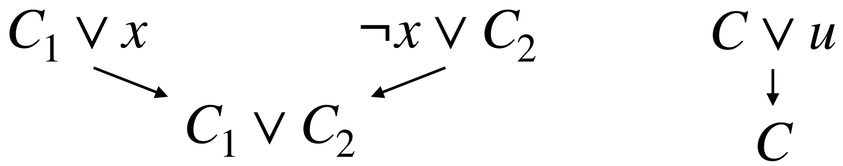
$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$



Q-Resolution



QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

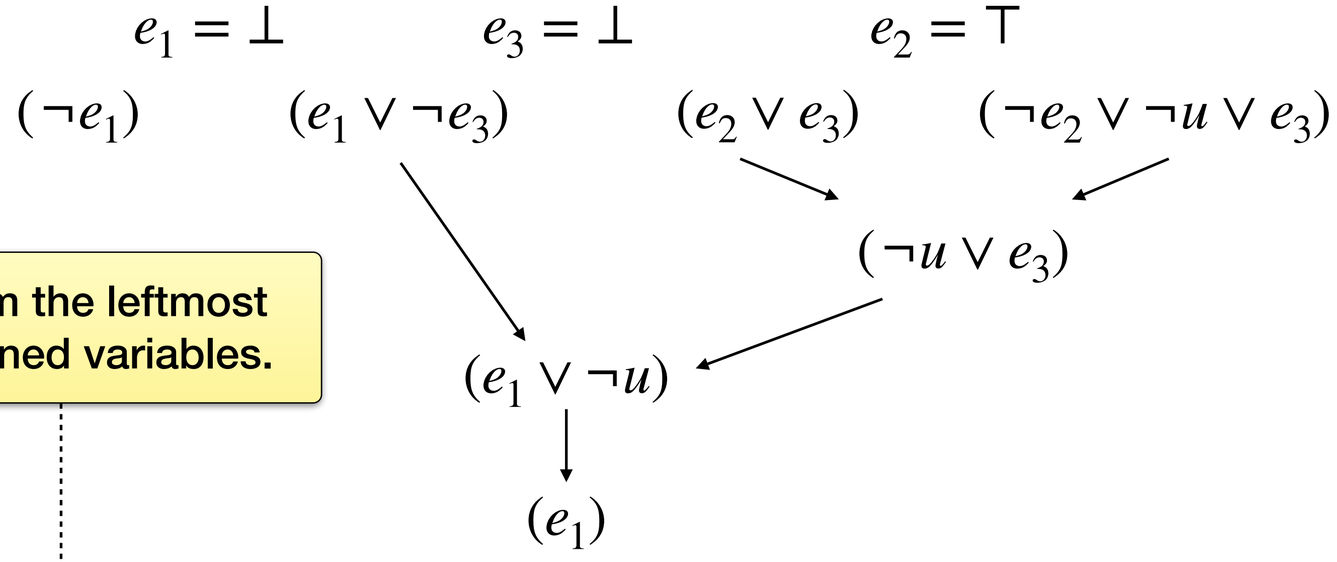
Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$



QCDCL with Propositional Unit Propagation

Propagate existential units.
Conflict at purely universal clauses.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Pick a variable from the leftmost block with unassigned variables.

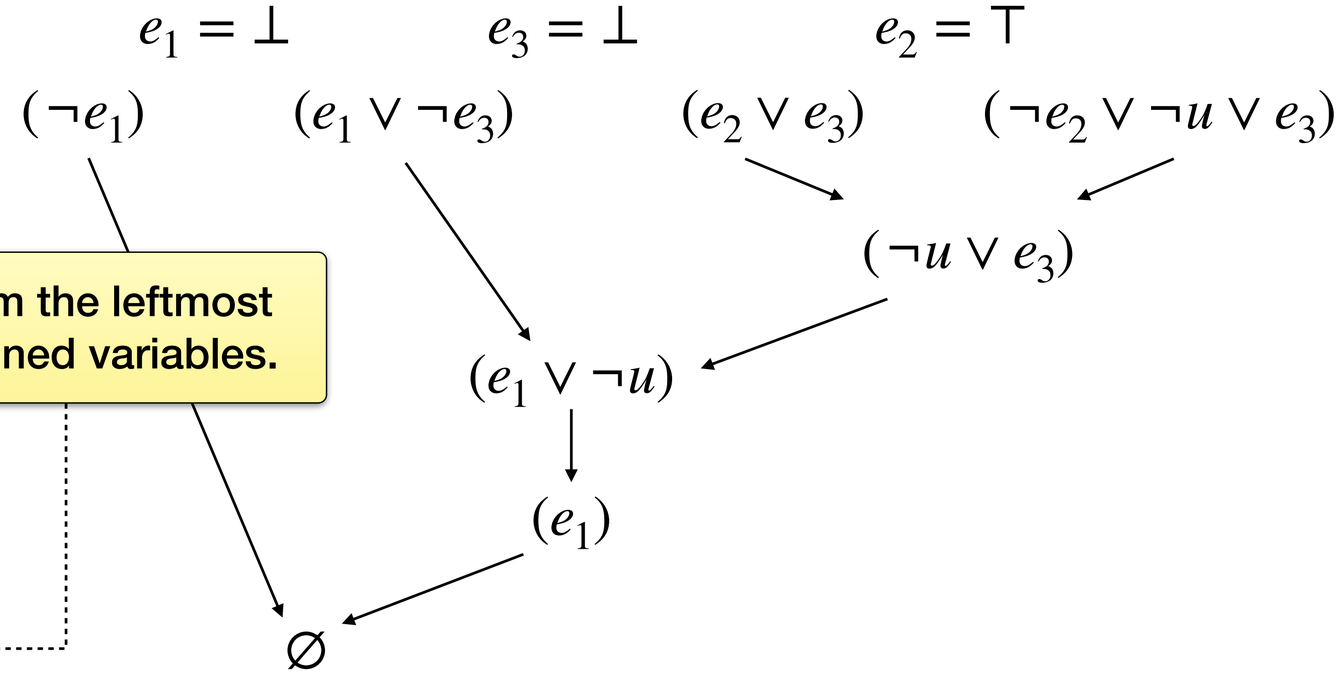
Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

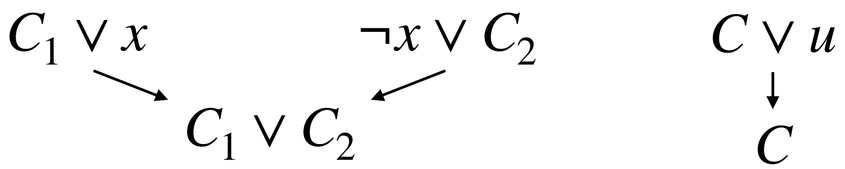
$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Resolution



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 \stackrel{d}{=} \bot$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

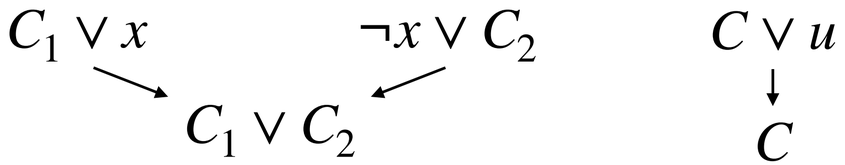
$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 \stackrel{d}{=} \bot$$

$$(u_1 \vee \neg e_1)$$

Q-Resolution



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$$

$$(u_1 \vee \neg e_1)$$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$$(u_1 \vee \neg e_1)$$

$u_2 \stackrel{d}{=} \bot$

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$

$(e_1 \vee u_2 \vee e_2)$

Q-Resolution

$C_1 \vee x \qquad \neg x \vee C_2 \qquad C \vee u$

$C_1 \vee C_2 \qquad C$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$$

$$(u_1 \vee \neg e_1)$$

$$u_2 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_1 \vee u_2 \vee e_2)$$

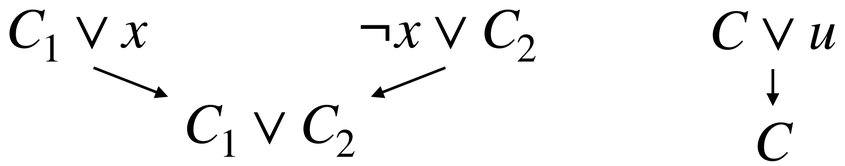
Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Resolution



$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$$

$$(u_1 \vee \neg e_1)$$

$$u_2 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_1 \vee u_2 \vee e_2)$$

$$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Resolution

$$\frac{C_1 \vee x \qquad \neg x \vee C_2}{C_1 \vee C_2} \qquad \frac{C \vee u}{C}$$

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot \qquad e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

$T_1 \wedge x$

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

“Model Generation”

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

$T_1 \wedge x \qquad \neg x \wedge T_2$

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot \qquad e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Q-Consensus

$T_1 \wedge x$ $\qquad$ $\neg x \wedge T_2$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Q-Consensus

$T_1 \wedge x$ $\qquad$ $\neg x \wedge T_2$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Q-Consensus

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

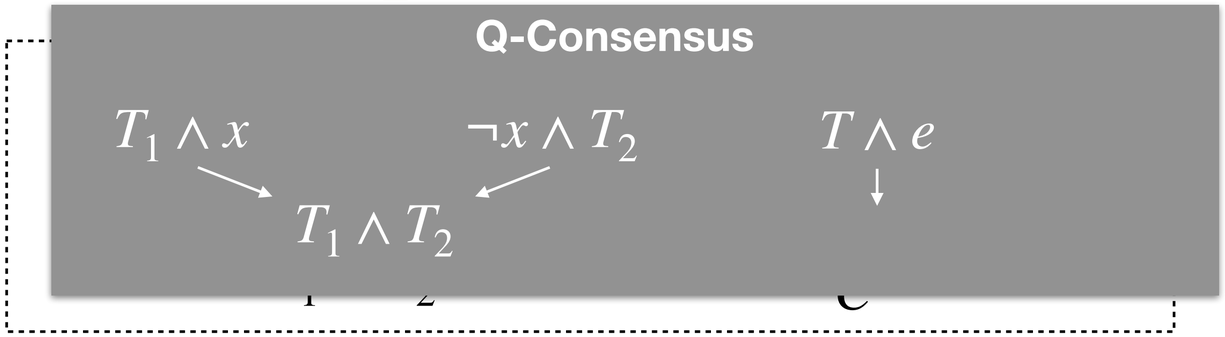
Q-Consensus

$T_1 \wedge x$ $\qquad$ $\neg x \wedge T_2$ $\qquad$ $T \wedge e$

$T_1 \wedge T_2$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```



$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\quad e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\quad e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

Q-Consensus

$T_1 \wedge x$ $\quad \neg x \wedge T_2$ $\quad T \wedge e$

$T_1 \wedge T_2$ $\quad T$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

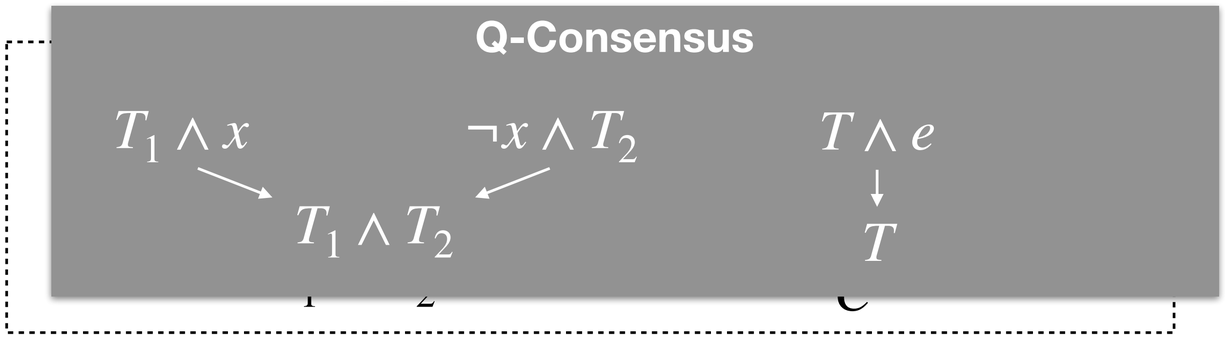
$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\qquad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$

$u_2 \stackrel{d}{=} \bot$ $\qquad$ $e_2 = \top$

$(e_1 \vee u_2 \vee e_2)$

“Model Generation”

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2 \wedge e_2)$

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2)$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

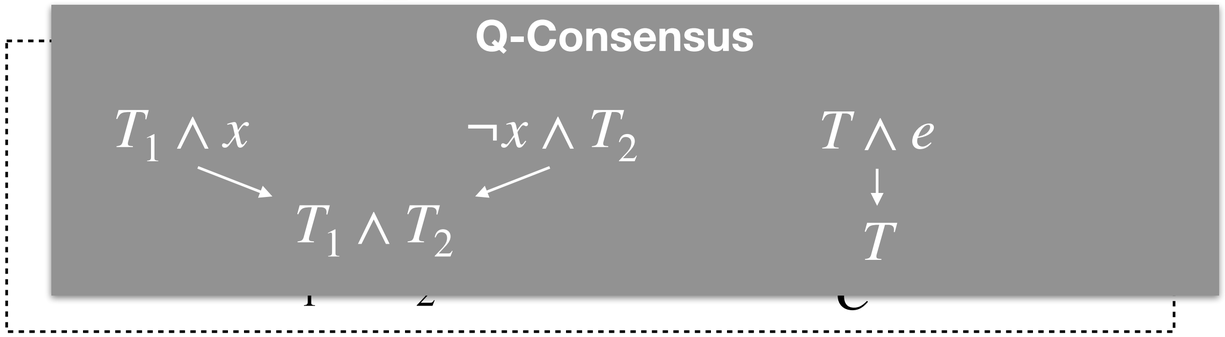
$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 \stackrel{d}{=} \bot \qquad e_1 = \bot$$

$$(u_1 \vee \neg e_1)$$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

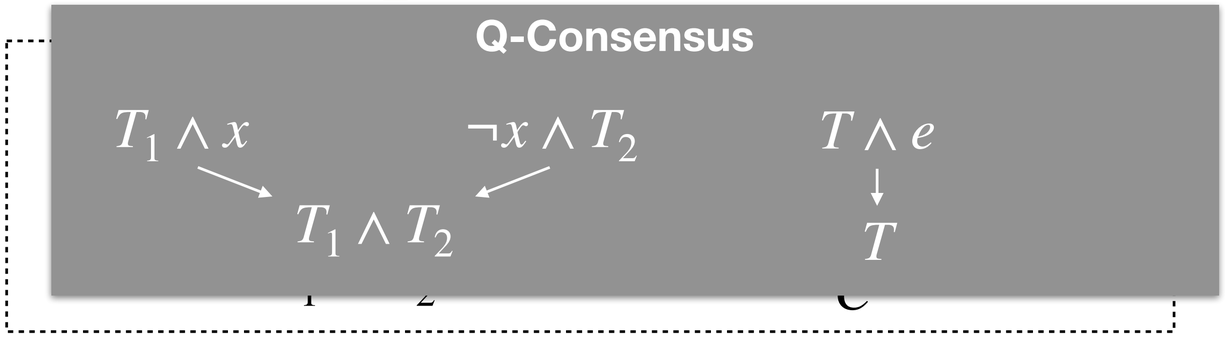
$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad$ $e_1 = \bot$

$(u_1 \vee \neg e_1)$ $\quad$ $(\neg u_1 \wedge \neg e_1 \wedge \neg u_2)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

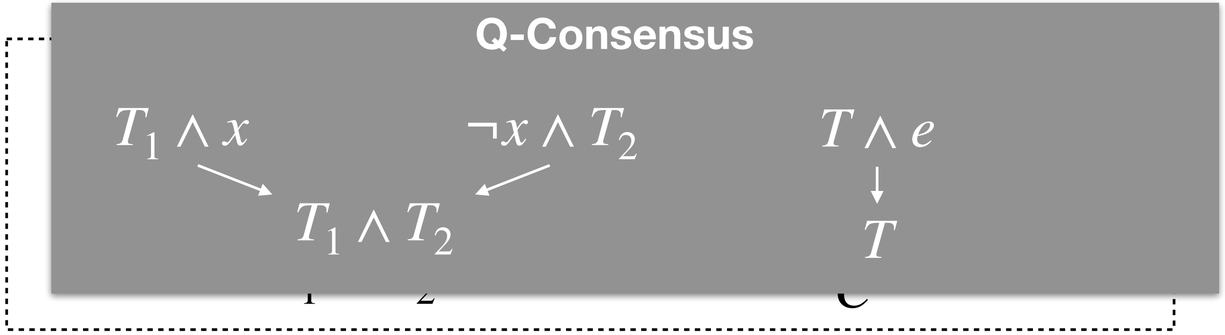
$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad e_1 = \bot$ $\quad u_2 = \top$

$(u_1 \vee \neg e_1)$ $\quad (\neg u_1 \wedge \neg e_1 \wedge \neg u_2)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

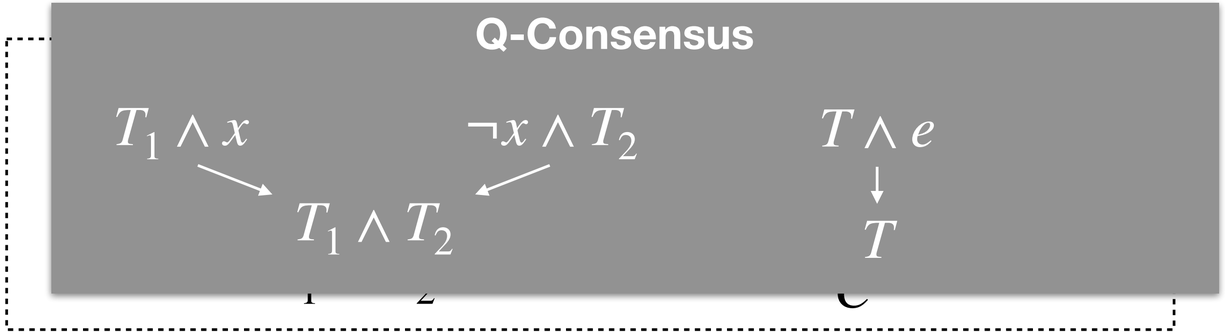
$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad e_1 = \bot$ $\quad u_2 = \top$

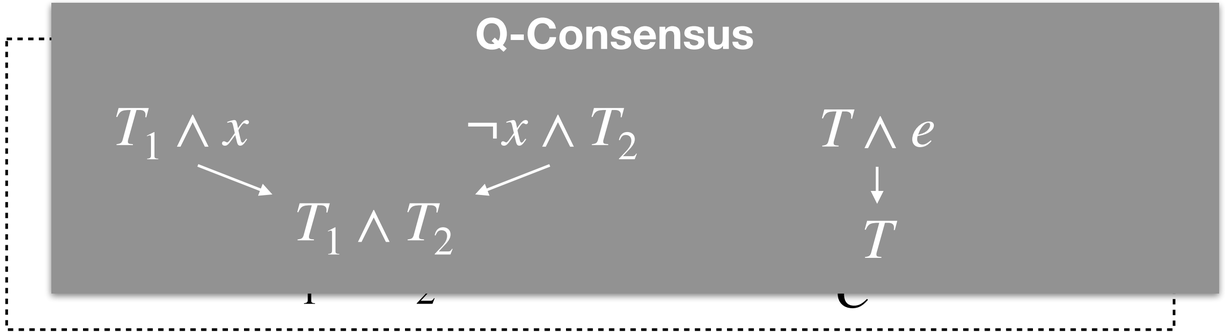
$(u_1 \vee \neg e_1)$ $\quad (\neg u_1 \wedge \neg e_1 \wedge \neg u_2)$ $\quad (\neg u_1 \wedge \neg e_1 \wedge u_2)$

Q-Consensus



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```



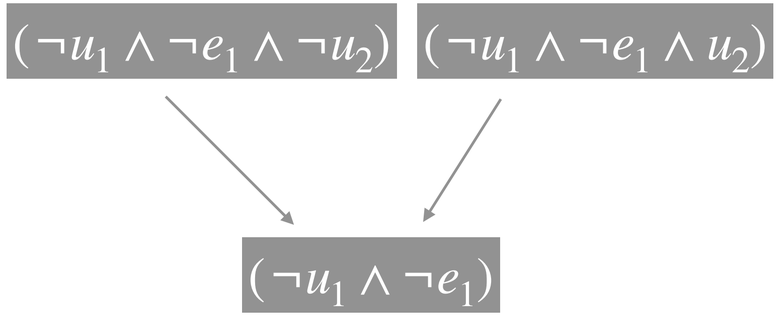
$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad e_1 = \bot$ $\quad u_2 = \top$

$(u_1 \vee \neg e_1)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 \stackrel{d}{=} \bot$ $\quad$ $e_1 = \bot$ $\quad$ $u_2 = \top$

$(u_1 \vee \neg e_1)$

$(\neg u_1 \wedge \neg e_1 \wedge \neg u_2)$ $\quad$ $(\neg u_1 \wedge \neg e_1 \wedge u_2)$

$(\neg u_1 \wedge \neg e_1)$

$(\neg u_1)$

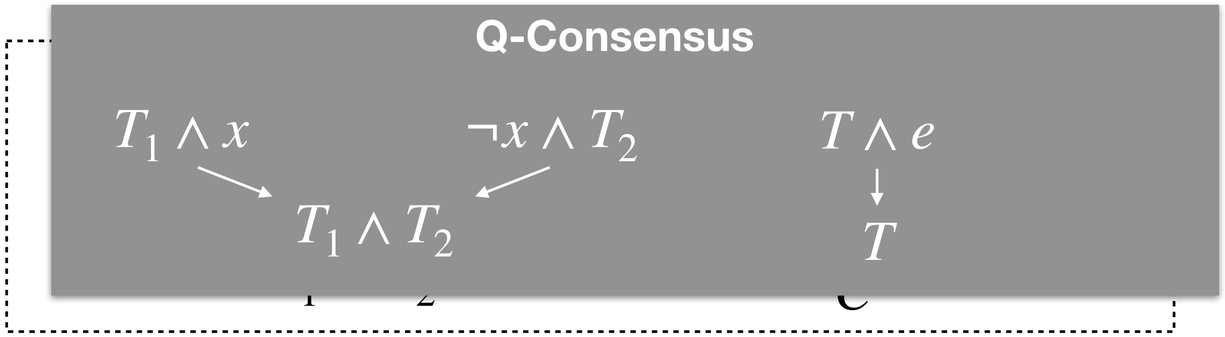
Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$



Learning From Satisfying Assignments

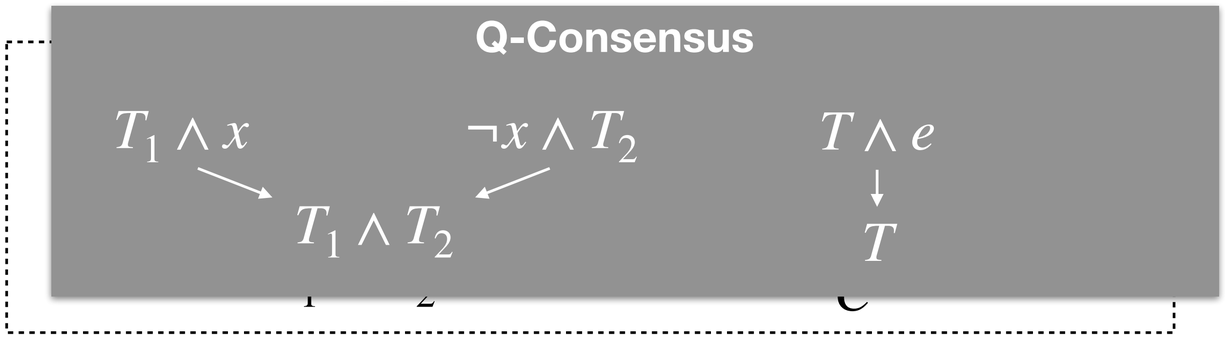
```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$(\neg u_1)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 = \top$$

$(\neg u_1)$

Q-Consensus

$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

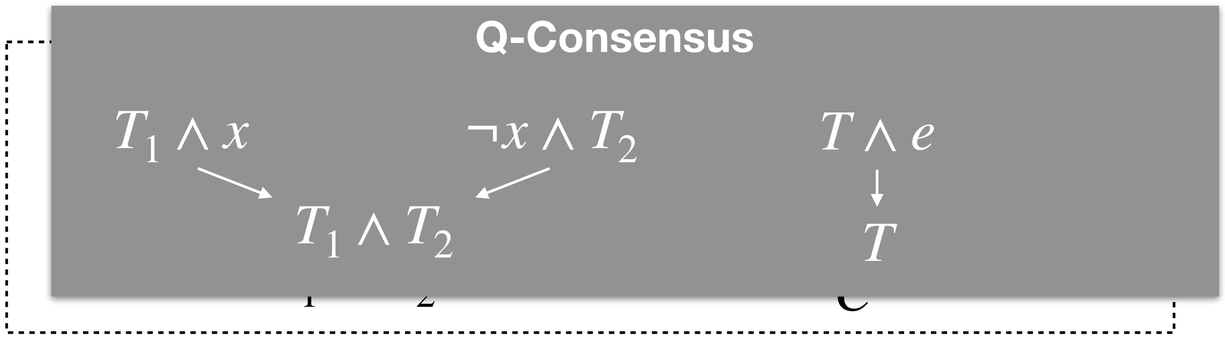
$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 = \top$$

$(\neg u_1)$ $\quad$ $(\neg u_1 \vee e_1)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $\quad$ $e_1 = \top$

$(\neg u_1)$ $\quad$ $(\neg u_1 \vee e_1)$

Q-Consensus

$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$$u_1 = \top \qquad e_1 = \top$$

$(\neg u_1)$ $\qquad (\neg u_1 \vee e_1)$

$$u_2 \stackrel{d}{=} \bot$$

Q-Consensus

$T_1 \wedge x$ $\qquad \neg x \wedge T_2$ $\qquad T \wedge e$

$T_1 \wedge T_2$ $\qquad T$

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $\quad$ $e_1 = \top$

$(\neg u_1)$ $\quad$ $(\neg u_1 \vee e_1)$

$u_2 \stackrel{d}{=} \bot$ $\quad$ $(u_1 \wedge e_1 \wedge \neg u_2)$

Q-Consensus

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $\quad$ $e_1 = \top$

$(\neg u_1)$ $\quad$ $(\neg u_1 \vee e_1)$

Q-Consensus

$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

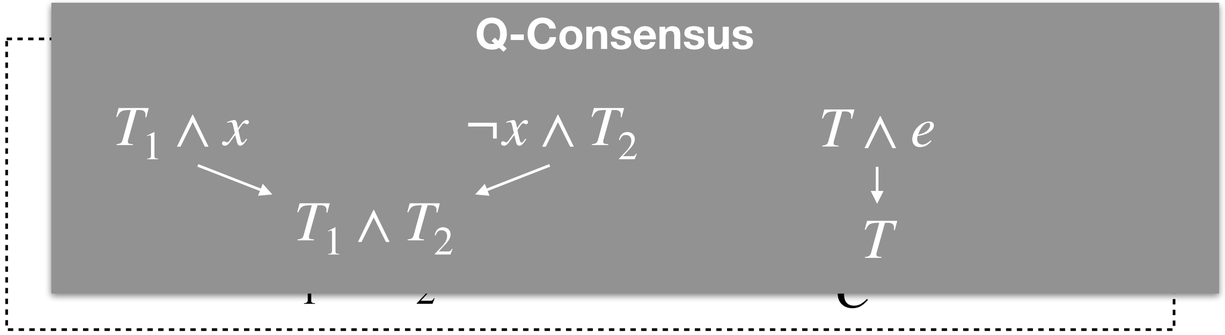
$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $e_1 = \top$

$(\neg u_1)$ $(\neg u_1 \vee e_1)$ $(u_1 \wedge e_1 \wedge \neg u_2)$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

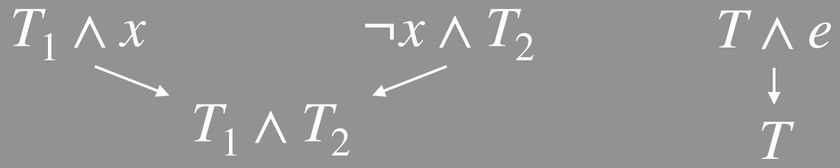
$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $(\neg u_1)$

$e_1 = \top$ $(\neg u_1 \vee e_1)$

$u_2 = \top$ $(u_1 \wedge e_1 \wedge \neg u_2)$

Q-Consensus



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $e_1 = \top$ $u_2 = \top$

$(\neg u_1)$ $(\neg u_1 \vee e_1)$ $(u_1 \wedge e_1 \wedge \neg u_2)$ $(\neg e_1 \vee \neg u_2 \vee \neg e_2)$

Q-Consensus

$T_1 \wedge x$ $\neg x \wedge T_2$ $T \wedge e$

$T_1 \wedge T_2$ T

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

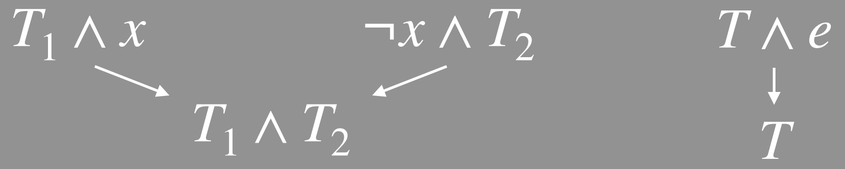
$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $e_1 = \top$ $u_2 = \top$ $e_2 = \bot$

$(\neg u_1)$ $(\neg u_1 \vee e_1)$ $(u_1 \wedge e_1 \wedge \neg u_2)$ $(\neg e_1 \vee \neg u_2 \vee \neg e_2)$

Q-Consensus



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

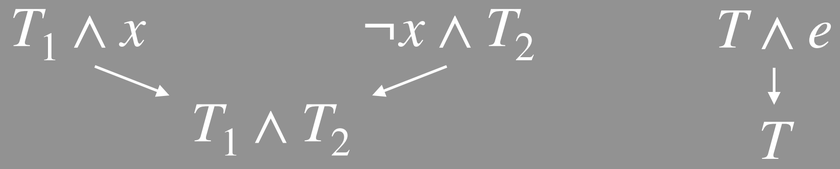
$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $e_1 = \top$ $u_2 = \top$ $e_2 = \bot$

$(\neg u_1)$ $(\neg u_1 \vee e_1)$ $(u_1 \wedge e_1 \wedge \neg u_2)$ $(\neg e_1 \vee \neg u_2 \vee \neg e_2)$

$(u_1 \wedge e_1 \wedge u_2 \wedge \neg e_2)$

Q-Consensus



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $e_1 = \top$ $u_2 = \top$ $e_2 = \bot$

$(\neg u_1)$ $(\neg u_1 \vee e_1)$ $(u_1 \wedge e_1 \wedge \neg u_2)$ $(\neg e_1 \vee \neg u_2 \vee \neg e_2)$

$(u_1 \wedge e_1 \wedge u_2 \wedge \neg e_2)$

$(u_1 \wedge e_1 \wedge u_2)$

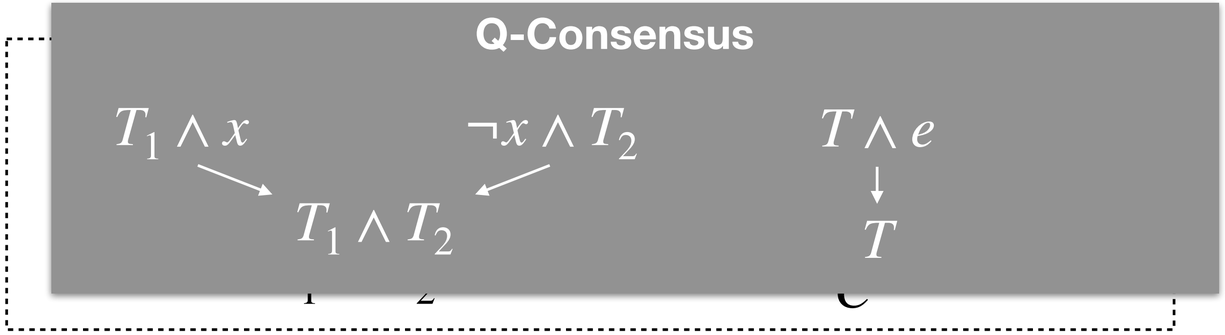
Q-Consensus

$T_1 \wedge x$ $\neg x \wedge T_2$ $T \wedge e$

$T_1 \wedge T_2$ T

Learning From Satisfying Assignments

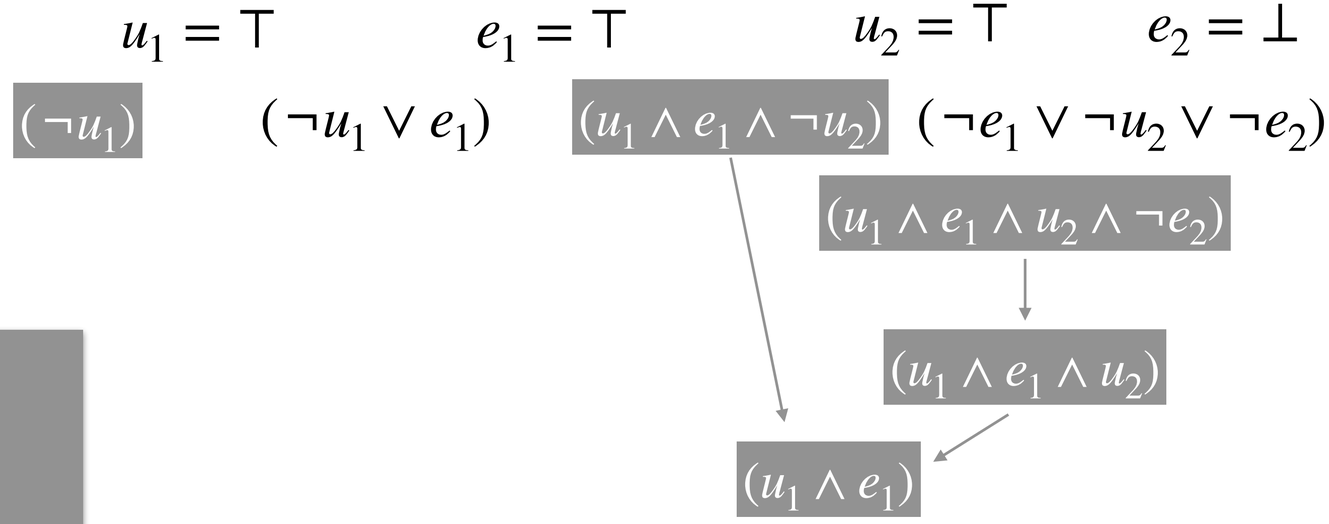
```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```



$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$



Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$

$u_1 = \top$ $\quad$ $e_1 = \top$ $\quad$ $u_2 = \top$ $\quad$ $e_2 = \bot$

$(\neg u_1)$ $\quad$ $(\neg u_1 \vee e_1)$ $\quad$ $(u_1 \wedge e_1 \wedge \neg u_2)$ $\quad$ $(\neg e_1 \vee \neg u_2 \vee \neg e_2)$

$(u_1 \wedge e_1 \wedge u_2 \wedge \neg e_2)$

$(u_1 \wedge e_1 \wedge u_2)$

$(u_1 \wedge e_1)$

(u_1)

Learning From Satisfying Assignments

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Q-Consensus

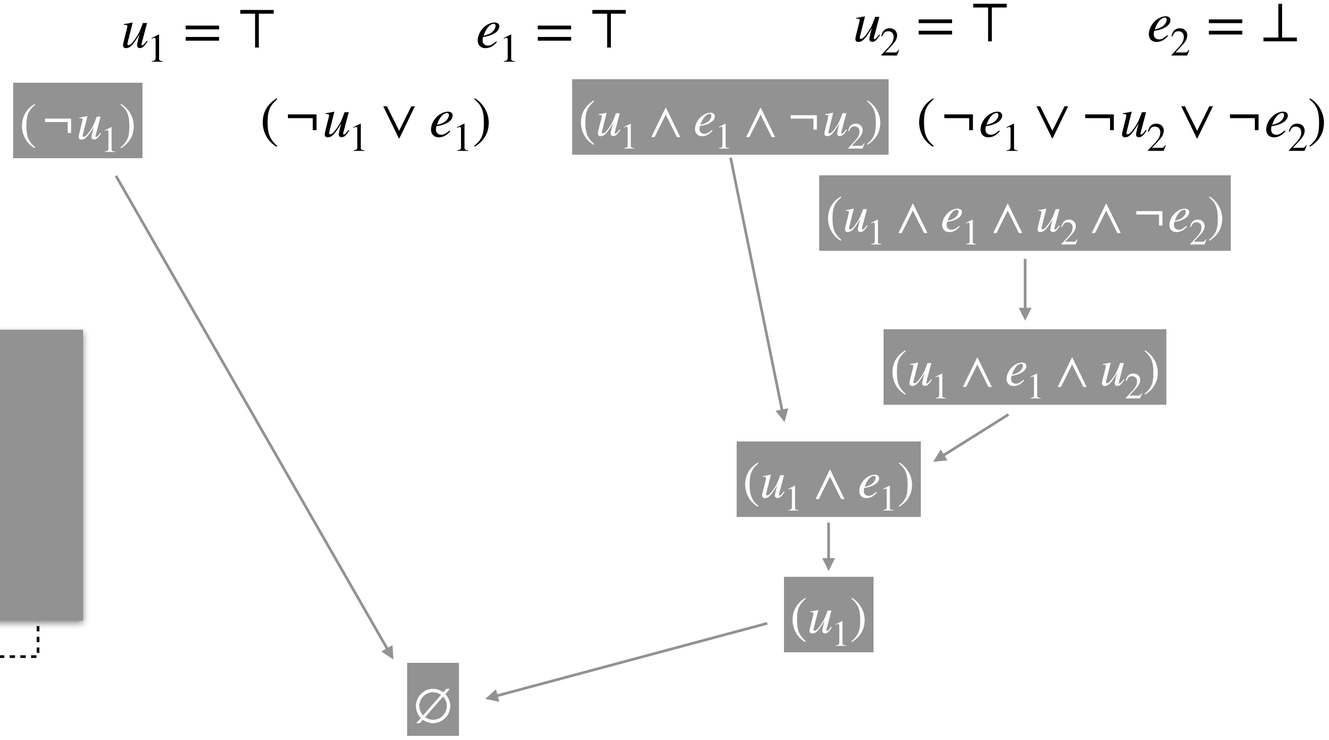
$T_1 \wedge x$ $\quad$ $\neg x \wedge T_2$ $\quad$ $T \wedge e$

$T_1 \wedge T_2$ $\quad$ T

$$\forall u_1 \exists e_1 \forall u_2 \exists e_2$$

$$(u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1)$$

$$(e_1 \vee u_2 \vee e_2) \wedge (\neg e_1 \vee \neg u_2 \vee \neg e_2)$$



Alternative to Model Generation

Alternative to Model Generation

- model generation leads to long initial terms

Alternative to Model Generation

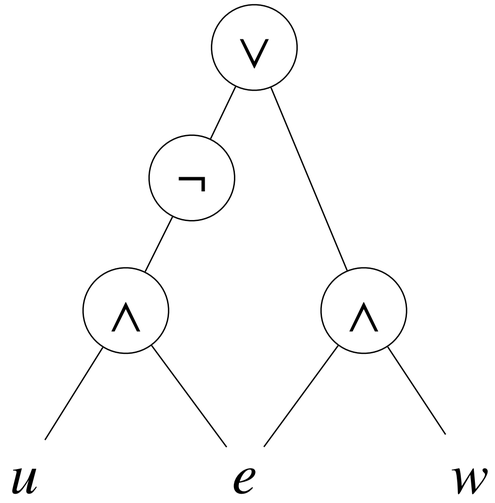
- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)

Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

Alternative to Model Generation

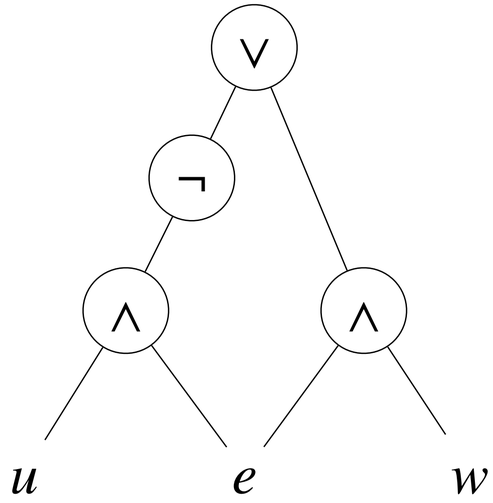
- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit



Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

$$\forall u \exists e \forall w$$

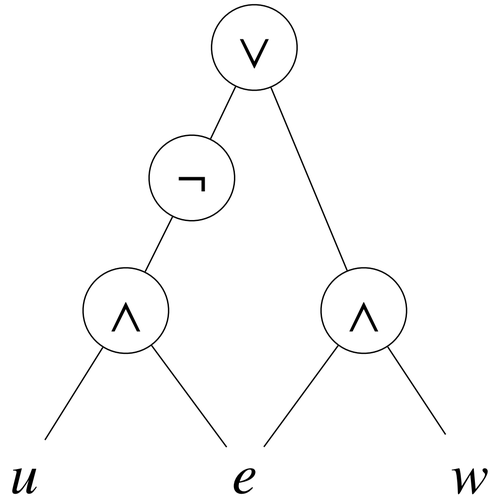


Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$\forall u \exists e \forall w$$



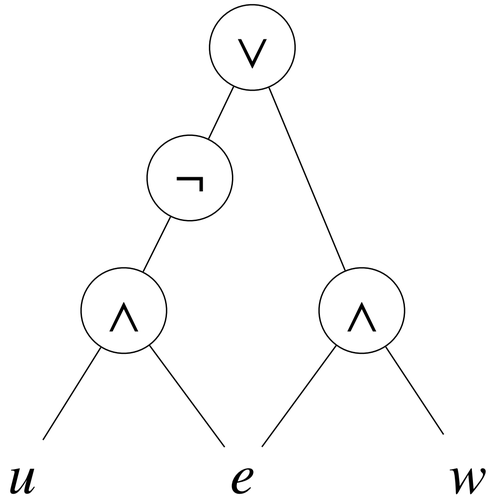
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



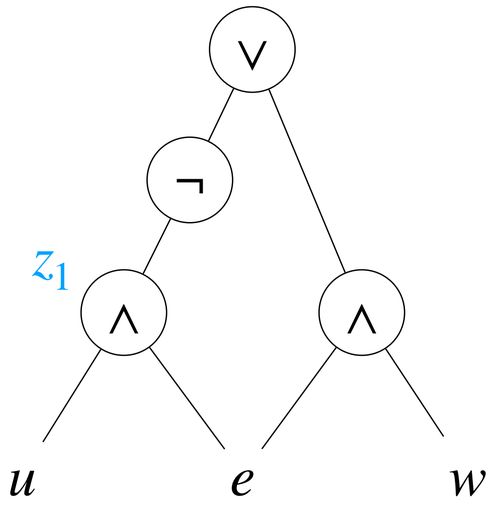
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



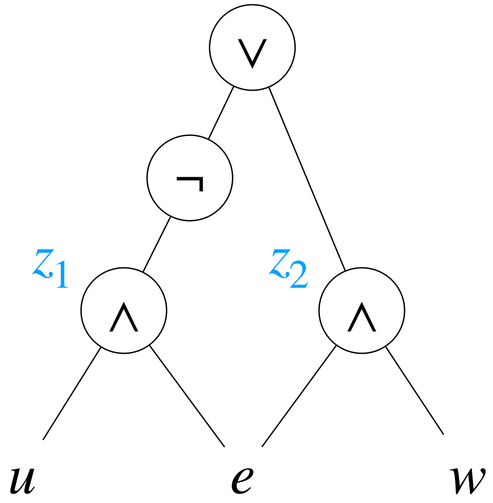
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



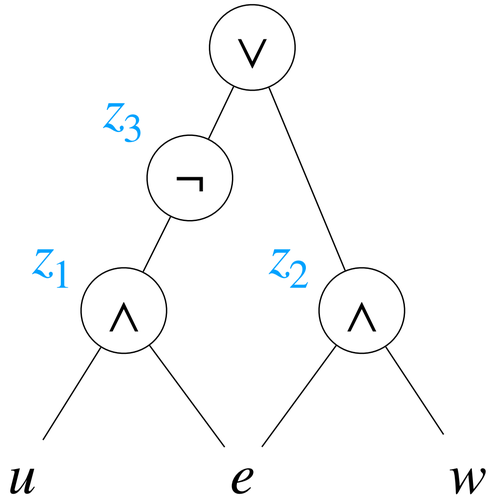
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



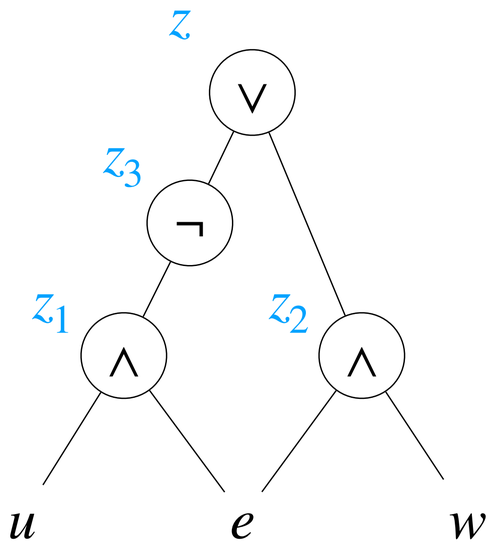
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



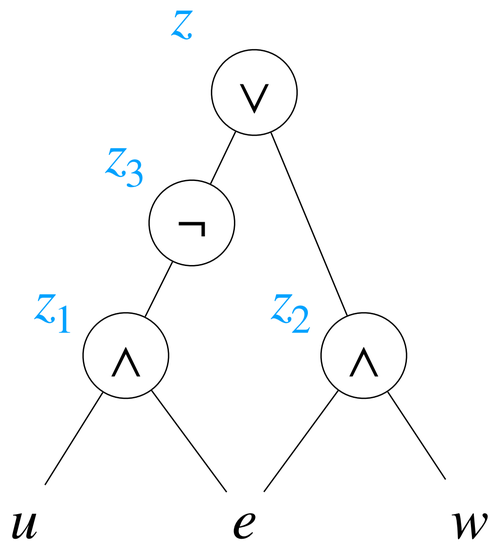
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$

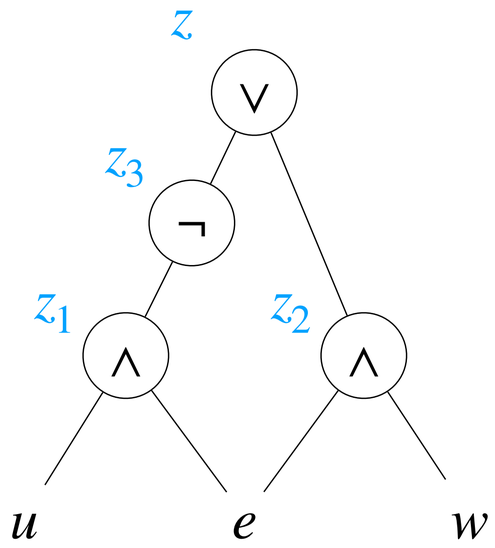
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



$\wedge$

$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$

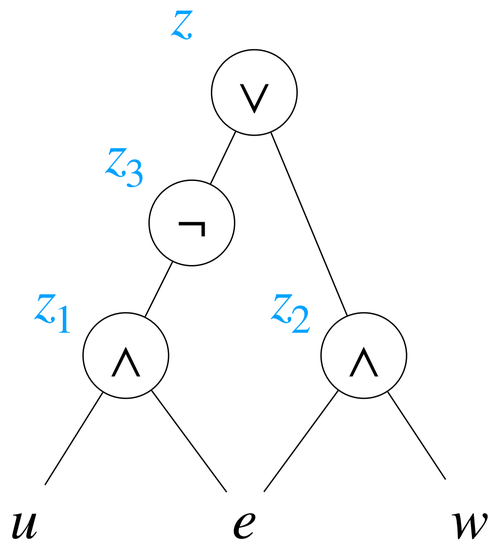
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

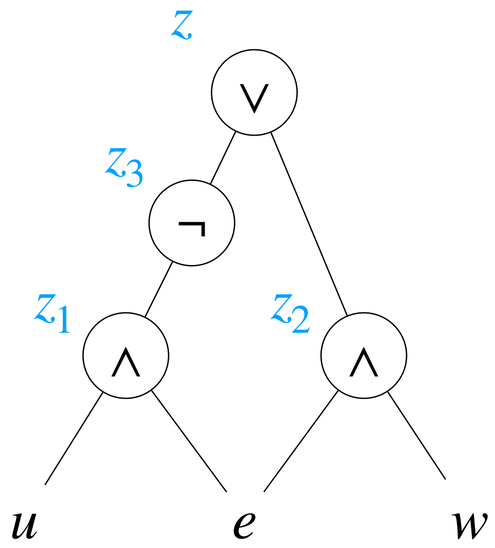
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$\forall u \exists e \forall w$

DNF



$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

Alternative to Model Generation

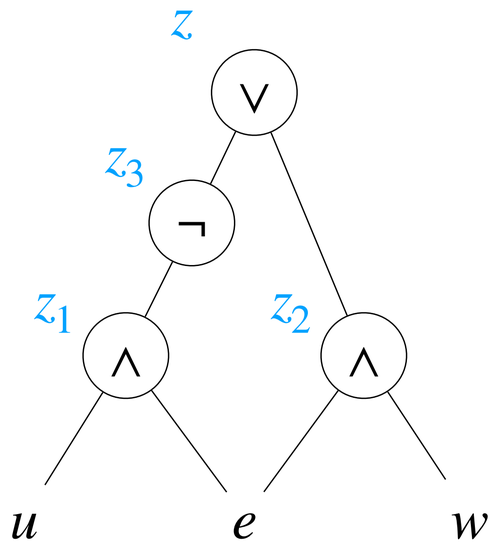
- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$\forall u \exists e \forall w$$

DNF

$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$



Alternative to Model Generation

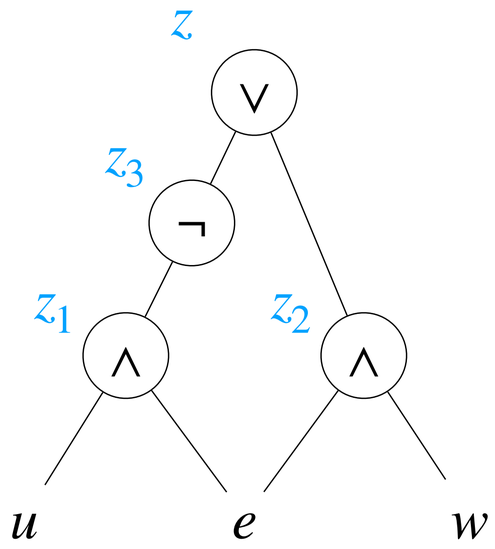
- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$\forall u \exists e \forall w$$

DNF

$$\begin{array}{c} \wedge \\ (z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3) \\ \wedge \\ (e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2) \\ \wedge \\ (u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1) \end{array}$$



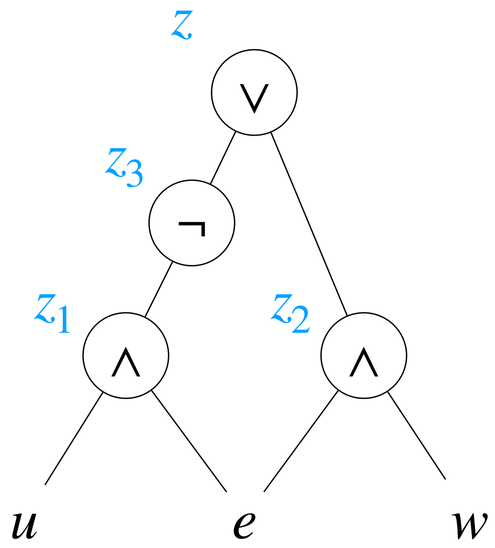
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

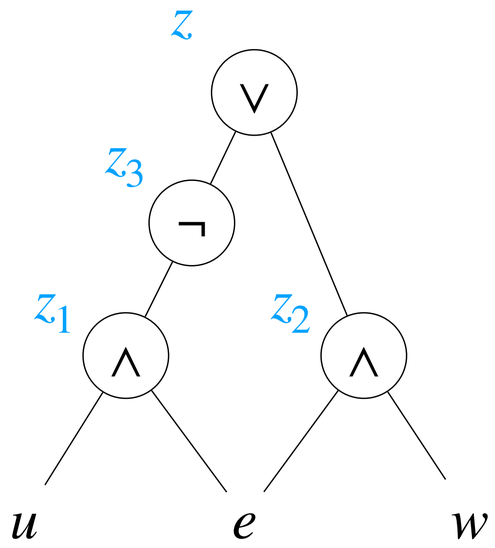
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$\bigwedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\bigwedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\bigwedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\bigwedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

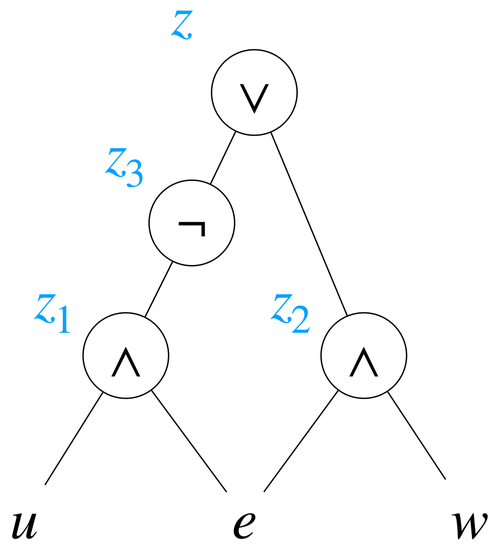
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

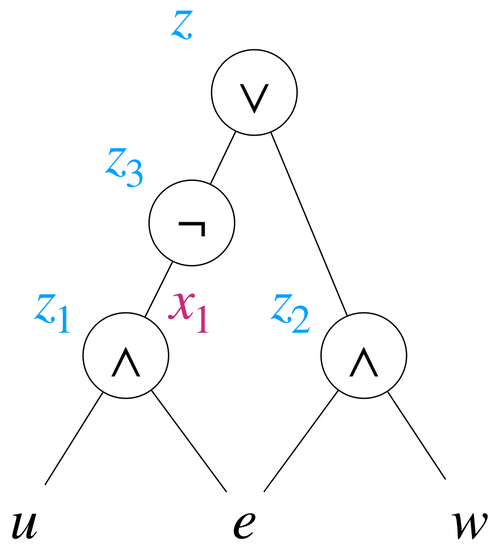
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

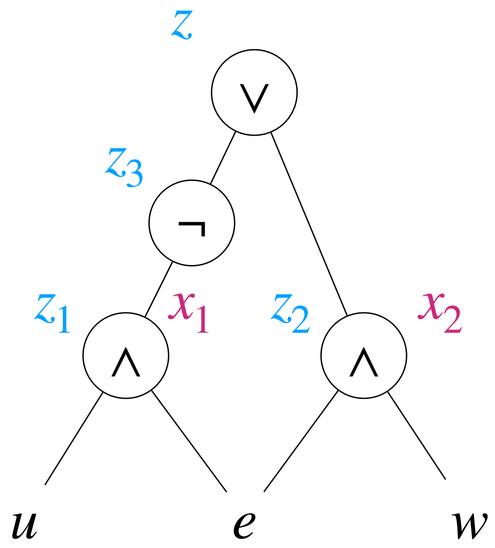
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

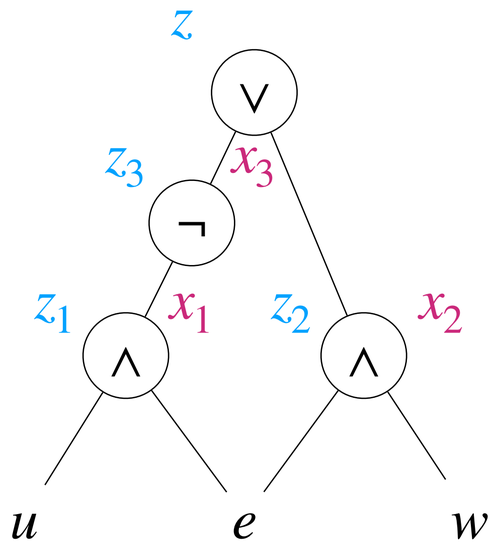
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

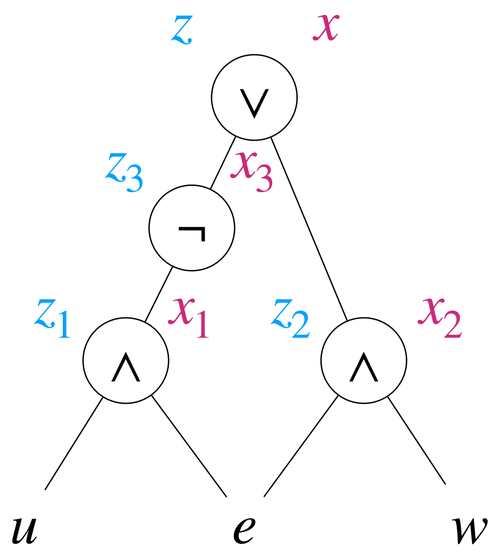
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

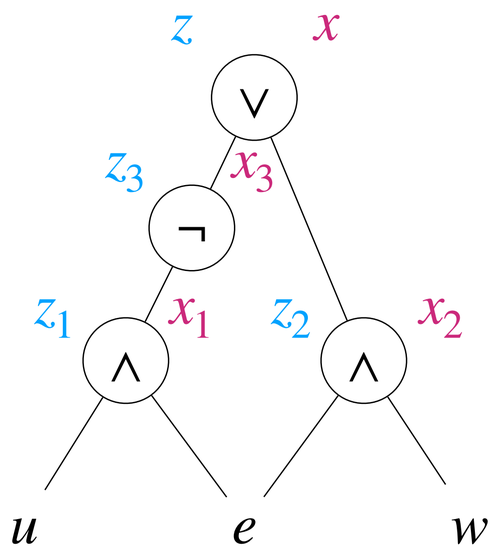
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

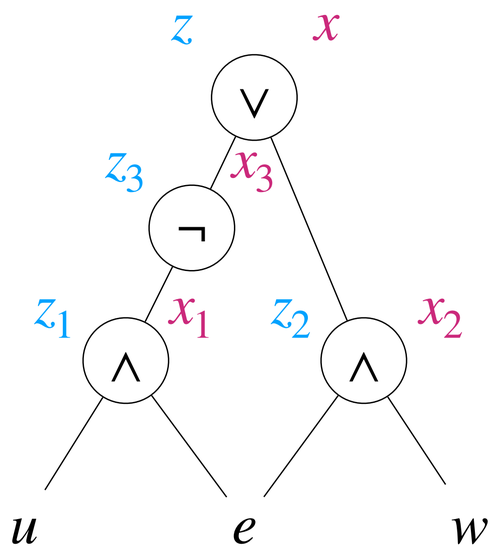
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

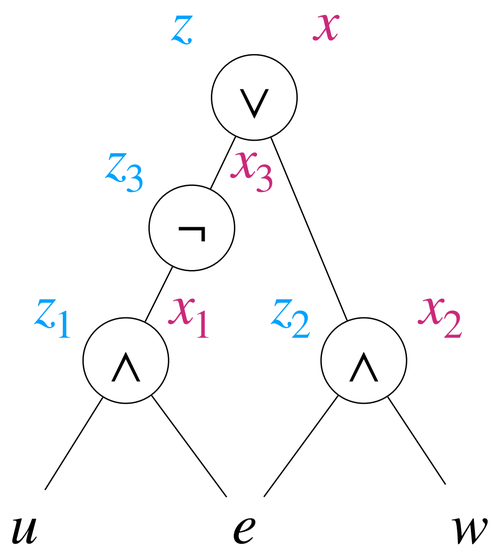
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

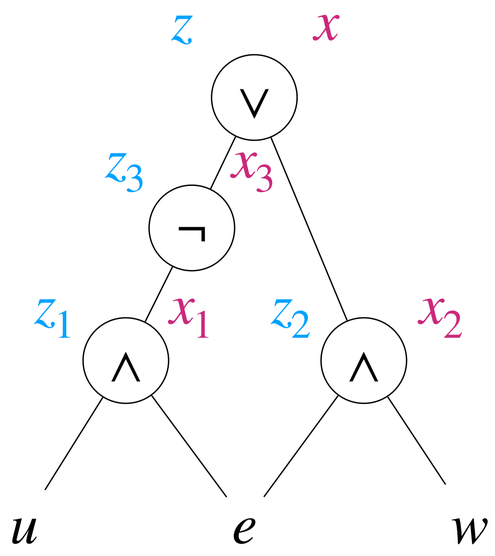
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

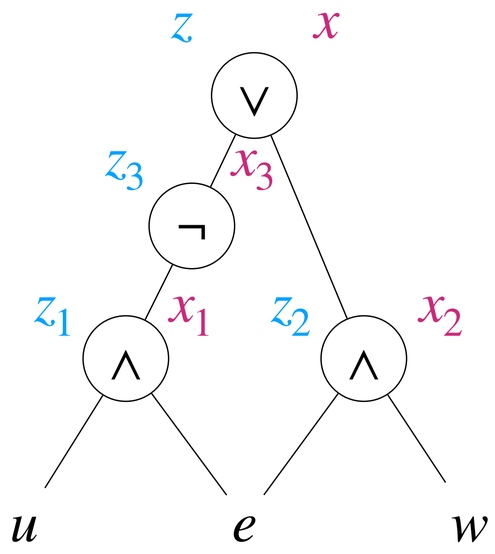
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

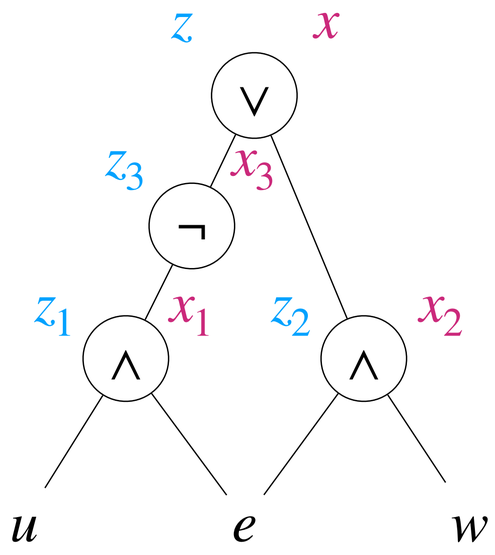
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$\vee$$
$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

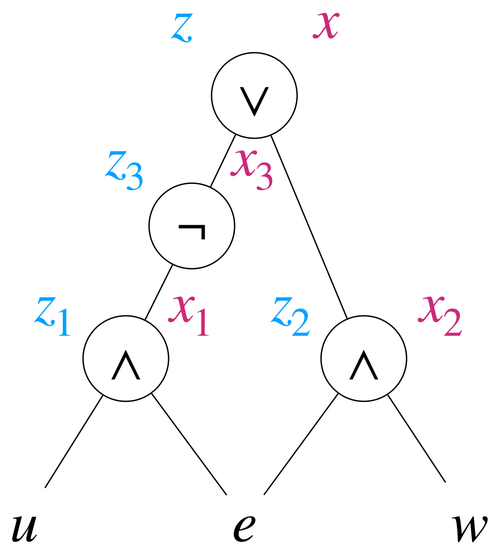
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$(x_3 \wedge \neg x) \vee (x_2 \wedge \neg x) \vee (\neg x_3 \vee \neg x_2 \wedge x)$$
$$\vee$$
$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

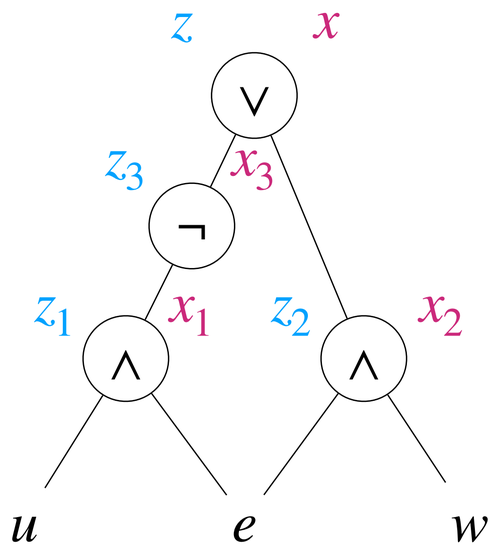
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$\vee$$
$$(x_3 \wedge \neg x) \vee (x_2 \wedge \neg x) \vee (\neg x_3 \vee \neg x_2 \wedge x)$$
$$\vee$$
$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

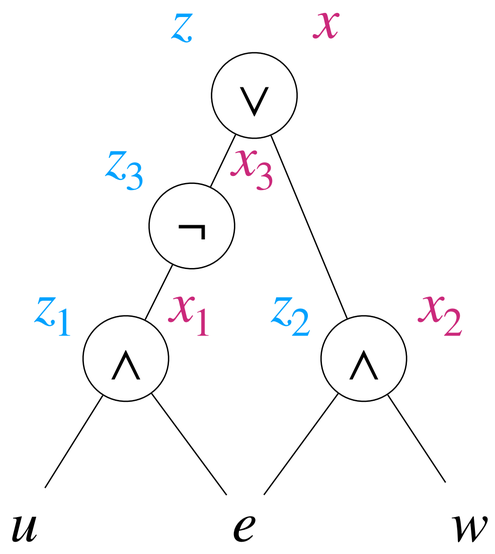
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$\forall u \exists e \forall w$



DNF

$$(x)$$
$$\vee$$
$$(x_3 \wedge \neg x) \vee (x_2 \wedge \neg x) \vee (\neg x_3 \vee \neg x_2 \wedge x)$$
$$\vee$$
$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

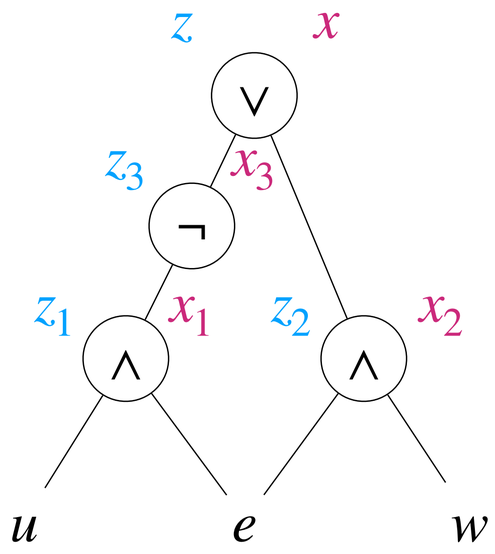
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

$$\forall u \exists e \forall w \; \exists z_1 \exists z_2 \exists z_3 \exists z$$

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$



DNF

$$(x)$$
$$\vee$$
$$(x_3 \wedge \neg x) \vee (x_2 \wedge \neg x) \vee (\neg x_3 \vee \neg x_2 \wedge x)$$
$$\vee$$
$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

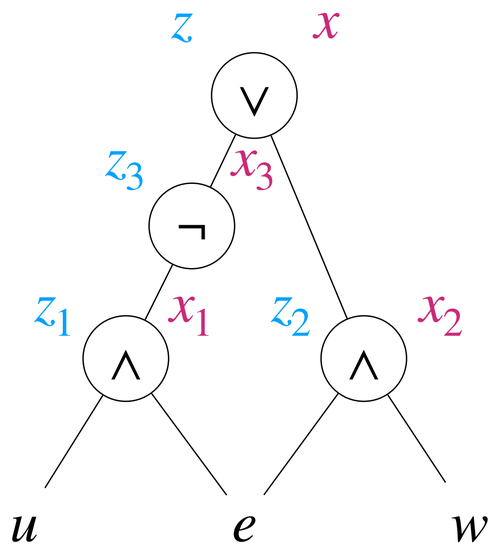
Alternative to Model Generation

- model generation leads to long initial terms
- finding short terms is a hard problem by itself (Hitting Set)
- it is typically better to generate both DNF and CNF from a circuit

CNF

$$(z)$$
$$\wedge$$
$$(\neg z_3 \vee z) \wedge (\neg z_2 \vee z) \wedge (z_3 \vee z_2 \vee \neg z)$$
$$\wedge$$
$$(z_1 \vee z_3) \wedge (\neg z_1 \vee \neg z_3)$$
$$\wedge$$
$$(e \vee \neg z_2) \wedge (w \vee \neg z_2) \wedge (\neg e \vee \neg w \vee z_2)$$
$$\wedge$$
$$(u \vee \neg z_1) \wedge (e \vee \neg z_1) \wedge (\neg u \vee \neg e \vee z_1)$$

$$\forall u \exists e \forall w \; \exists z_1 \exists z_2 \exists z_3 \exists z \; \forall x_1 \forall x_2 \forall x_3 \forall x$$



DNF

$$(x)$$
$$\vee$$
$$(x_3 \wedge \neg x) \vee (x_2 \wedge \neg x) \vee (\neg x_3 \vee \neg x_2 \wedge x)$$
$$\vee$$
$$(\neg x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3)$$
$$\vee$$
$$(\neg e \wedge x_2) \vee (\neg w \vee x_2) \vee (e \wedge w \wedge \neg x_2)$$
$$\vee$$
$$(\neg u \wedge x_1) \vee (\neg e \vee x_1) \vee (u \wedge e \wedge \neg x_1)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot$$

$$(e_2 \vee e_3)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

not “asserting”

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Dependency Schemes

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

not “asserting”

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Dependency Schemes Lonsing, Biere 2010

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \quad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

not “asserting”

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Dependency Schemes Lonsing, Biere 2010

Dependency Learning

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$e_3 \stackrel{d}{=} \bot$ $\quad$ $e_2 = \top$

$(e_2 \vee e_3)$ $\quad$ $(\neg e_2 \vee \neg u \vee e_3)$

$(\neg u \vee e_3)$

not “asserting”

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

not “asserting”

Dependency Schemes Lonsing, Biere 2010

Dependency Learning Peitl, Slivovsky, Szeider 2019

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

not “asserting”

Dependency Schemes Lonsing, Biere 2010

Dependency Learning Peitl, Slivovsky, Szeider 2019

QCDCL without Asserting Clauses

Quantified CDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

with out-of-order decisions

$$\exists e_1, e_2 \forall u \exists e_3, e_4$$

$$(e_1 \vee \neg e_3) \wedge (\neg e_1 \vee \neg e_4) \wedge (u \vee e_4) \wedge$$

$$(e_2 \vee e_3) \wedge (\neg e_2 \vee \neg u \vee e_3)$$

$$e_3 \stackrel{d}{=} \bot \qquad e_2 = \top$$

$$(e_2 \vee e_3) \qquad (\neg e_2 \vee \neg u \vee e_3)$$

$$(\neg u \vee e_3)$$

not “asserting”

Dependency Schemes Lonsing, Biere 2010

Dependency Learning Peitl, Slivovsky, Szeider 2019

QCDCL without Asserting Clauses Böhm, Peitl, Beyersdorff 2024

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

More Flavors of QCDCL

Apply universal reduction.
Propagate existential units.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

More Flavors of QCDCL

Apply universal reduction.
Propagate existential units.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clau
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Long-Distance Q-Resolution

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clau
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Apply universal reduction.
Propagate existential units.

Long-Distance Q-Resolution

Zhang, Malik 2002

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clau
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Apply universal reduction.
Propagate existential units.

Long-Distance Q-Resolution

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

More Flavors of QCDCL

Apply universal reduction.
Propagate existential units.

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(claus
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Apply universal reduction.
Propagate existential units.

Q-Resolution

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(claus
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Apply universal reduction.
Propagate existential units.

Q-Resolution

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

Lonsing, Egly, Van Gelder 2013

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

Lonsing, Egly, Van Gelder 2013

More Flavors of QCDCL

Propagate all units (including universals).

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clause_term)
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

Lonsing, Egly, Van Gelder 2013

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clau
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Propagate all units (including universals).

QU-Resolution

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

Lonsing, Egly, Van Gelder 2013

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clau
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Propagate all units (including universals).

QU-Resolution

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

Lonsing, Egly, Van Gelder 2013

Slivovsky 2022

More Flavors of QCDCL

```
def QCDCL():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause_term, bt_level = analyze(conflict)
      if clause_term == []:
        return is_term(clau
      attach(clause_term)
      backtrack(bt_level)
    elif allAssigned():
      conflict = model_generation()
      goto conflict_analysis
    else:
      decide_variable()
```

Propagate all units (including universals).

QU-Resolution

Zhang, Malik 2002

Egly, Widl, Lonsing 2013

Lonsing, Egly, Van Gelder 2013

Slivovsky 2022

TODO switch between propagation and reduction of universal variables

Challenges for QCDCL

Challenges for QCDCL

QCDCL often gets stuck learning long terms

Challenges for QCDCL

QCDCL often gets stuck learning long terms

generate terms in a smarter way

Challenges for QCDCL

QCDCL often gets stuck learning long terms

generate terms in a smarter way

lift inprocessing to QCDCL

Challenges for QCDCL

QCDCL often gets stuck learning long terms

generate terms in a smarter way

lift inprocessing to QCDCL

more freedom in decision order can be detrimental with current heuristics

Challenges for QCDCL

QCDCL often gets stuck learning long terms

generate terms in a smarter way

lift inprocessing to QCDCL

more freedom in decision order can be detrimental with current heuristics

design QBF-specific decision heuristics

3. Counterexample-Guided Abstraction Refinement (CEGAR)

Shannon Expansion

Shannon Expansion

$$\forall u \Phi$$

Shannon Expansion

$$\forall u \Phi \quad \equiv$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\ ]$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\ ] \ \wedge$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv \quad \bigwedge_{\tau \in 2^U}$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv \quad \bigwedge_{\tau \in 2^U} \quad \exists E\, \varphi[\tau]$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv \quad \bigwedge_{\tau \in 2^U} \quad \exists E\, \varphi[\tau]$$

$$\bigwedge_{\tau \in 2^U}$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv \quad \bigwedge_{\tau \in 2^U} \quad \exists E\, \varphi[\tau]$$

$$\bigwedge_{\tau \in 2^U} \quad \varphi[\tau][E \leftarrow E^\tau]$$

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv \quad \bigwedge_{\tau \in 2^U} \exists E\, \varphi[\tau]$$

$$\bigwedge_{\tau \in 2^U} \varphi[\tau][E \leftarrow E^\tau]$$

propositional

Shannon Expansion

$$\forall u \Phi \quad \equiv \quad \Phi[u \leftarrow \bot\,] \;\wedge\; \Phi[u \leftarrow \top\,]$$

$$\forall U \exists E\, \varphi \quad \equiv \quad \bigwedge_{\tau \in 2^U} \quad \exists E\, \varphi[\tau]$$

$$\bigwedge_{\tau \in 2^U} \quad \varphi[\tau][E \leftarrow E^\tau]$$

propositional

already a **small subset** of clauses may be unsatisfiable

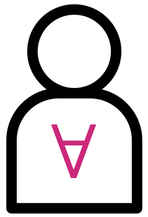
2QBF as a Game

2QBF as a Game

$\forall U \exists E \, \varphi$

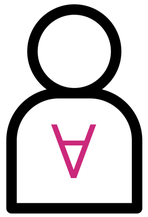
2QBF as a Game

$\forall U \exists E \, \varphi$



2QBF as a Game

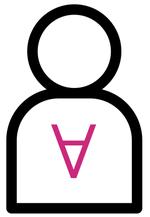
$\forall U \exists E \, \varphi$

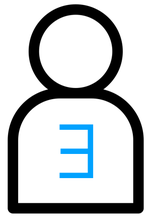


$U = \tau$

2QBF as a Game

$\forall U \exists E \, \varphi$

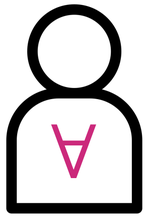


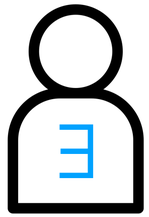


$U = \tau$

2QBF as a Game

$$\forall U \exists E \, \varphi$$



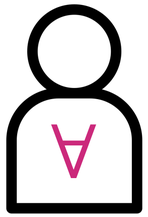


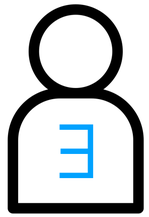
$U = \tau$

$E = \sigma$

2QBF as a Game

$\forall U \exists E \, \varphi$





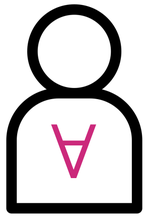
∃ wins if $\tau \cup \sigma \vDash \varphi$

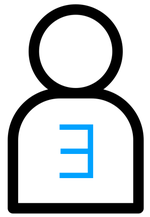
$U = \tau$

$E = \sigma$

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

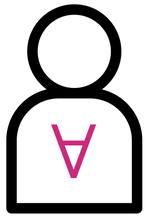
$U = \tau$

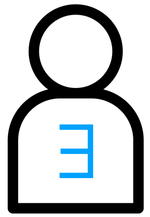
$U = \tau_1$

$E = \sigma$

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

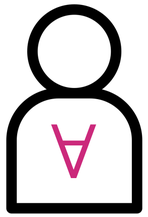
$E = \sigma$

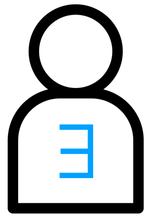
$U = \tau_1$

$\varphi[\tau_1]$ satisfiable?

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

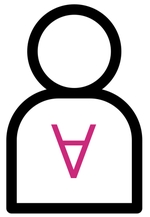
$E = \sigma$

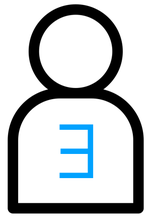
$U = \tau_1$

$\varphi[\tau_1]$ satisfiable? ⇢ **No**

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

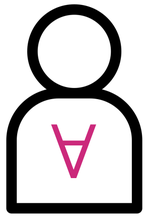
$U = \tau_1$

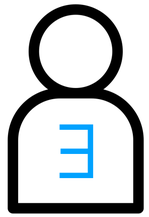
$\varphi[\tau_1]$ satisfiable? ⇢ **No**

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

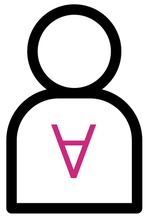
$U = \tau_1$

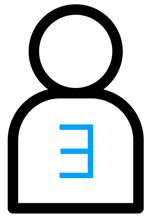
Yes ⟵ $\varphi[\tau_1]$ satisfiable? ⟶ **No**

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

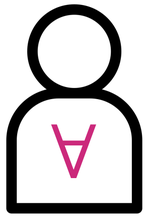
Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

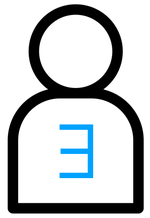
$U = \tau_2$

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$ $E = \sigma$

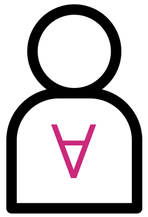
$U = \tau_1$ **Yes** ← $\varphi[\tau_1]$ satisfiable? → **No**

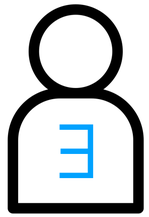
$U = \tau_2$ $\varphi[\tau_2]$ satisfiable?

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

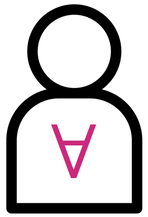
$U = \tau_2$

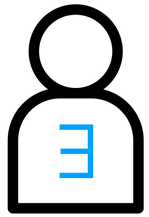
$\varphi[\tau_2]$ satisfiable? → **No**

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

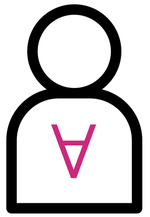
$U = \tau_2$

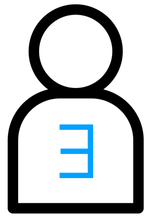
Yes ← $\varphi[\tau_2]$ satisfiable? → **No**

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$

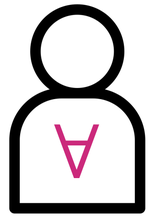
Yes ← $\varphi[\tau_2]$ satisfiable? → **No**

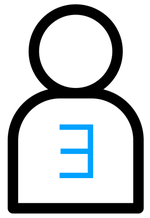
…

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$

Yes ← $\varphi[\tau_2]$ satisfiable? → **No**

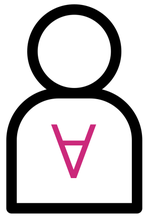
…

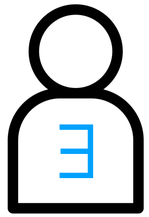
…

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$

Yes ← $\varphi[\tau_2]$ satisfiable? → **No**

…

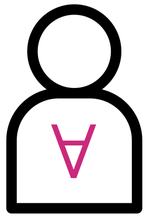
…

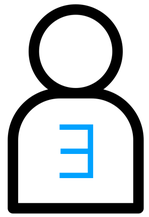
$U = \tau_N$

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$

Yes ← $\varphi[\tau_2]$ satisfiable? → **No**

$N = 2^{|U|}$

…

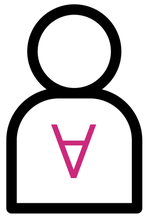
…

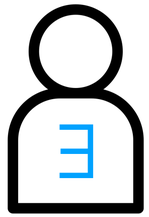
$U = \tau_N$

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$

Yes ← $\varphi[\tau_2]$ satisfiable? → **No**

$N = 2^{|U|}$

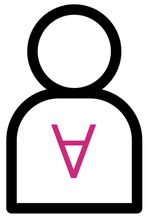
$\ldots$ $\quad \ldots$

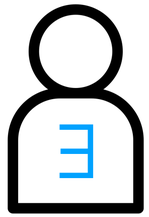
$U = \tau_N$ $\quad \varphi[\tau_N]$ satisfiable?

`return False`

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$ — **Yes** ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$ — **Yes** ← $\varphi[\tau_2]$ satisfiable? → **No**

$N = 2^{|U|}$

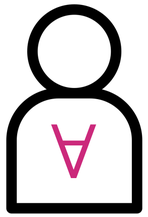
$\ldots$ $\quad \ldots$

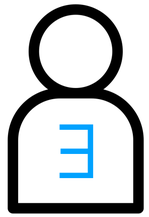
$U = \tau_N$ — $\varphi[\tau_N]$ satisfiable? → **No**

```
return False
```

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$ — **Yes** ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$ — **Yes** ← $\varphi[\tau_2]$ satisfiable? → **No**

$N = 2^{|U|}$

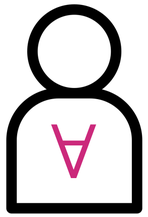
$\ldots$ $\qquad$ $\ldots$

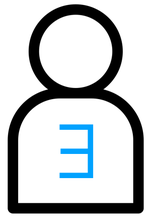
$U = \tau_N$ — **Yes** ← $\varphi[\tau_N]$ satisfiable? → **No**

`return False`

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \models \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$ — **Yes** ← $\varphi[\tau_1]$ satisfiable? → **No**

$U = \tau_2$ — **Yes** ← $\varphi[\tau_2]$ satisfiable? → **No**

$N = 2^{|U|}$

…

…

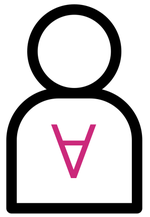
$U = \tau_N$ — **Yes** ← $\varphi[\tau_N]$ satisfiable? → **No**

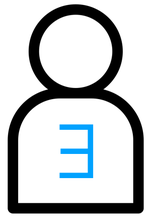
`return True`

`return False`

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

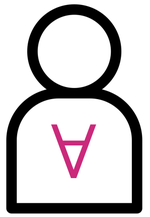
$U = \tau_1$

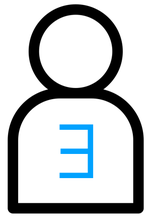
$\varphi[\tau_1]$ satisfiable?

$U = \tau_2$

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

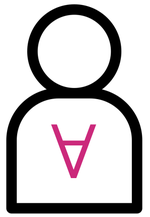
$U = \tau_1$

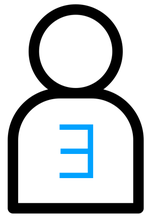
Yes ⇠ $\varphi[\tau_1]$ satisfiable?

$U = \tau_2$

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

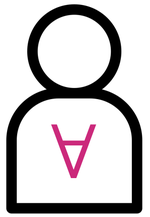
Yes ⇠ $\varphi[\tau_1]$ satisfiable?

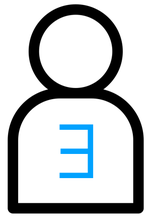
$E = \sigma_1$

$U = \tau_2$

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$ $E = \sigma$

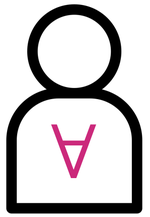
$U = \tau_1$ **Yes** ⇠ $\varphi[\tau_1]$ satisfiable?

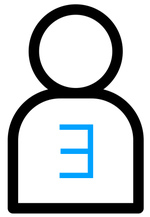
$E = \sigma_1$

$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable?

$E = \sigma_1$

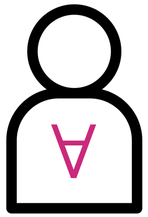
$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

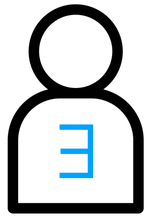
$\cdots$

$U = \tau_k$

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$ $E = \sigma$

$U = \tau_1$ **Yes** ← $\varphi[\tau_1]$ satisfiable?

$E = \sigma_1$

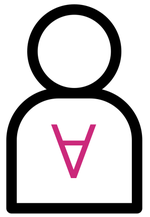
$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

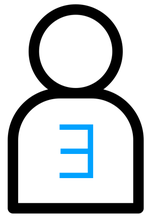
…

$U = \tau_k$ $\tau_k \cup \sigma_1$ falsifies φ

2QBF as a Game

$\forall U \exists E\, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$ $E = \sigma$

$U = \tau_1$ **Yes** ← $\varphi[\tau_1]$ satisfiable?

$\sigma_1, \ldots, \sigma_i$

$E = \sigma_1$

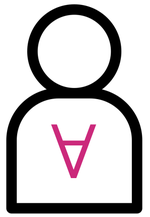
$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

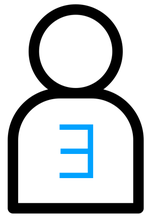
$\ldots$

$U = \tau_k$ $\tau_k \cup \sigma_1$ falsifies φ

2QBF as a Game

$\forall U \exists E\, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$ $E = \sigma$

$U = \tau_1$ **Yes** ← $\varphi[\tau_1]$ satisfiable?

$E = \sigma_1$

$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

$\sigma_1, \ldots, \sigma_i$

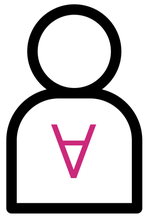
set $U = \tau_{i+1}$

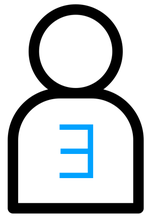
$\ldots$

$U = \tau_k$ $\tau_k \cup \sigma_1$ falsifies φ

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable?

$E = \sigma_1$

$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

$\sigma_1, \ldots, \sigma_i$

set $U = \tau_{i+1}$

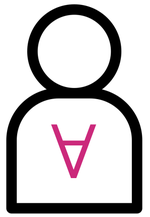
$\tau_{i+1} \vDash \neg\varphi[\sigma_1] \wedge \ldots \wedge \neg\varphi[\sigma_i]$

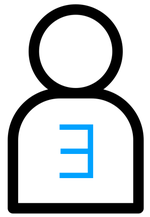
…

$U = \tau_k$ $\tau_k \cup \sigma_1$ falsifies φ

2QBF as a Game

$\forall U \exists E \, \varphi$





∃ wins if $\tau \cup \sigma \vDash \varphi$

$U = \tau$

$E = \sigma$

$U = \tau_1$

Yes ← $\varphi[\tau_1]$ satisfiable?

$E = \sigma_1$

$U = \tau_2$ if $\tau_2 \cup \sigma_1$ satisfies φ

…

$U = \tau_k$ $\tau_k \cup \sigma_1$ falsifies φ

$\sigma_1, \ldots, \sigma_i$

set $U = \tau_{i+1}$

$\tau_{i+1} \vDash \neg\varphi[\sigma_1] \wedge \ldots \wedge \neg\varphi[\sigma_i]$

“abstraction”

CEGAR Expansion

Janota, Marques-Silva 2011

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top$ $\tau = u_1 \wedge u_2$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top$ $\tau = u_1 \wedge u_2$ $\sigma = e$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top$ $\quad \tau = u_1 \wedge u_2$ $\quad \sigma = e$ $\quad \varphi[\sigma] = (u_1) \wedge (u_2)$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top$ $\quad \tau = u_1 \wedge u_2$ $\quad \sigma = e$ $\quad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top$ $\quad \tau = u_1 \wedge u_2$ $\quad \sigma = e$ $\quad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2$ $\quad \tau = \neg u_1 \wedge \neg u_2$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top \qquad \tau = u_1 \wedge u_2 \qquad \sigma = e \qquad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2 \qquad \tau = \neg u_1 \wedge \neg u_2 \qquad \sigma = \neg e$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top \qquad \tau = u_1 \wedge u_2 \qquad \sigma = e \qquad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2 \qquad \tau = \neg u_1 \wedge \neg u_2 \qquad \sigma = \neg e \qquad \varphi[\sigma] = (\neg u_1) \wedge (\neg u_2)$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top \qquad \tau = u_1 \wedge u_2 \qquad \sigma = e \qquad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2 \qquad \tau = \neg u_1 \wedge \neg u_2 \qquad \sigma = \neg e \qquad \varphi[\sigma] = (\neg u_1) \wedge (\neg u_2)$

$$\mathscr{A} = (\neg u_1 \vee \neg u_2) \wedge (u_1 \vee u_2)$$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top \qquad \tau = u_1 \wedge u_2 \qquad \sigma = e \qquad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2 \qquad \tau = \neg u_1 \wedge \neg u_2 \qquad \sigma = \neg e \qquad \varphi[\sigma] = (\neg u_1) \wedge (\neg u_2)$

$\mathscr{A} = (\neg u_1 \vee \neg u_2) \wedge (u_1 \vee u_2) \qquad \tau = u_1 \wedge \neg u_2$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top \qquad \tau = u_1 \wedge u_2 \qquad \sigma = e \qquad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2 \qquad \tau = \neg u_1 \wedge \neg u_2 \qquad \sigma = \neg e \qquad \varphi[\sigma] = (\neg u_1) \wedge (\neg u_2)$

$\mathscr{A} = (\neg u_1 \vee \neg u_2) \wedge (u_1 \vee u_2) \qquad \tau = u_1 \wedge \neg u_2 \qquad \varphi[\tau] = (e) \wedge (\neg e)$

CEGAR Expansion

Janota, Marques-Silva 2011

```
def CEGAR_Exp():
  𝒜 = ⊤
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    mat_sat, σ = SAT(φ[τ])
    if not mat_sat:
      return False
    # Refine
    𝒜 = 𝒜 ∧ ¬φ[σ]
```

$$\forall u_1, u_2 \exists e \, (u_1 \vee \neg e) \wedge (\neg u_1 \vee e) \wedge (u_2 \vee \neg e) \wedge (\neg u_2 \vee e)$$

$\mathscr{A} = \top \qquad \tau = u_1 \wedge u_2 \qquad \sigma = e \qquad \varphi[\sigma] = (u_1) \wedge (u_2)$

$\mathscr{A} = \neg u_1 \vee \neg u_2 \qquad \tau = \neg u_1 \wedge \neg u_2 \qquad \sigma = \neg e \qquad \varphi[\sigma] = (\neg u_1) \wedge (\neg u_2)$

$\mathscr{A} = (\neg u_1 \vee \neg u_2) \wedge (u_1 \vee u_2) \qquad \tau = u_1 \wedge \neg u_2 \qquad \varphi[\tau] = (e) \wedge (\neg e)$

`mat_sat = False`

An Adversarial Example

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$$u_1 \wedge u_2 \wedge \ldots \wedge u_n$$

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\qquad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$$u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

$$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$$

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$$u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

$$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad \neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$$u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

$$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad \neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

$$\ldots$$

An Adversarial Example

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$$u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

$$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad \neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$$

$$\ldots$$

needs 2^n iterations

CEGAR Expansion with ML

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$$u_1 \wedge u_2 \wedge \ldots \wedge u_n$$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\qquad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\qquad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

...

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n \qquad \neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

...

try to **predict** e_i **from** u_i

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

...

$u_1 \leftrightarrow e_1$

try to **predict** e_i **from** u_i

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

...

$u_1 \leftrightarrow e_1 \quad u_2 \leftrightarrow e_2$

try to **predict** e_i **from** u_i

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\ldots$

$u_1 \leftrightarrow e_1 \quad u_2 \leftrightarrow e_2 \quad \ldots$

try to **predict** e_i **from** u_i

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\ldots$

$u_1 \leftrightarrow e_1 \quad u_2 \leftrightarrow e_2 \quad \ldots \quad u_n \leftrightarrow e_n$

try to **predict** e_i **from** u_i

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\ldots$

$u_1 \leftrightarrow e_1 \quad u_2 \leftrightarrow e_2 \quad \ldots \quad u_n \leftrightarrow e_n$

try to **predict** e_i **from** u_i

$$\mathcal{A} = \mathcal{A} \wedge \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i \wedge \neg\varphi$$

CEGAR Expansion with ML

Janota 2018

$$\forall u_1 \ldots \forall u_n \exists e_1 \ldots \exists e_n \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i$$

$u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\neg u_1 \wedge u_2 \wedge \ldots \wedge u_n$ $\quad$ $\neg e_1 \wedge e_2 \wedge \ldots \wedge e_n$

$\ldots$

$u_1 \leftrightarrow e_1 \quad u_2 \leftrightarrow e_2 \quad \ldots \quad u_n \leftrightarrow e_n$

try to **predict e_i from u_i**

$$\mathscr{A} = \mathscr{A} \wedge \bigwedge_{i=1}^{n} u_i \leftrightarrow e_i \wedge \neg\varphi$$

unsatisfiable

More on CEGAR Expansion

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

Janota, Klieber, Marques-Silva, Clarke 2016

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

work on the underlying proof system ∀Exp+Res

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

work on the underlying proof system ∀Exp+Res

Janota, Marques-Silva 2015

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

work on the underlying proof system ∀Exp+Res

Janota, Marques-Silva 2015

Q-resolution proofs can be exponentially shorter

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

work on the underlying proof system ∀Exp+Res

Janota, Marques-Silva 2015

Q-resolution proofs can be exponentially shorter

Beyersdorff, Janota, Chew 2019

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

exponential separation
in the other direction

work on the underlying proof system ∀Exp+Res

Janota, Marques-Silva 2015

Q-resolution proofs can
be exponentially shorter

Beyersdorff, Janota, Chew 2019

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

exponential separation in the other direction

work on the underlying proof system ∀Exp+Res

Janota, Marques-Silva 2015

Q-resolution proofs can be exponentially shorter

Beyersdorff, Janota, Chew 2019

Beyersdorff, Chew, Clymo, Mahajan 2019

More on CEGAR Expansion

can be generalized to arbitrary quantifier prefixes

recursive

Janota, Klieber, Marques-Silva, Clarke 2016

Bloem, Braud-Santoni, Hadzic, Egly, Lonsing, Seidl 2021

without recursion

exponential separation in the other direction

work on the underlying proof system ∀Exp+Res

Janota, Marques-Silva 2015

Q-resolution proofs can be exponentially shorter

Beyersdorff, Janota, Chew 2019

Beyersdorff, Chew, Clymo, Mahajan 2019

∀Exp+Res p-simulates Q-resolution for bounded alternations

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

$$C_1$$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 . \underbrace{(x_1 \vee \neg x_2 \vee y_1)}_{C_1} \wedge \underbrace{(x_2 \vee \neg y_1 \vee \neg y_2)}_{C_2} \wedge (y_1 \vee y_2)$$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

$$\tau = \neg x_1 \wedge \neg x_2$$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

$\tau = \neg x_1 \wedge \neg x_2$ $(\neg y_1 \vee \neg y_2)$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

$\tau = \neg x_1 \wedge \neg x_2$ $(\neg y_1 \vee \neg y_2)$ $(y_1 \vee y_2)$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

$\tau = \neg x_1 \wedge \neg x_2$

$(\neg y_1 \vee \neg y_2)$ $(y_1 \vee y_2)$

$\forall$ “**selects**” C_2 and C_3

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

$\tau = \neg x_1 \wedge \neg x_2$

$(\neg y_1 \vee \neg y_2)$ $(y_1 \vee y_2)$

$\forall$ “**selects**” C_2 and C_3

$\sigma = \neg y_1 \wedge y_2$

Clause Selection

$$\forall x_1, x_2 \exists y_1, y_2 \,.\, (x_1 \vee \neg x_2 \vee y_1) \wedge (x_2 \vee \neg y_1 \vee \neg y_2) \wedge (y_1 \vee y_2)$$

C_1 C_2 C_3

$\tau = \neg x_1 \wedge \neg x_2$

$(\neg y_1 \vee \neg y_2)$ $(y_1 \vee y_2)$

$\forall$ “**selects**” C_2 and C_3

$\sigma = \neg y_1 \wedge y_2$

$\forall$ must select at least one clause not satisfied by σ

CEGAR with Clause Selection

CEGAR with Clause Selection

Janota, Marques-Silva 2015

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
```

$\mathscr{A} = \{(\neg s_C \vee \neg \ell) \mid C \in \varphi, \ell \in C, var(\ell) \in U\}$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$\forall u_1 \forall u_2 \exists e_1 \exists e_2$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

s_1

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

s_1 s_2

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underset{s_1}{(u_1 \vee e_1)} \wedge \underset{s_2}{(u_2 \vee e_2)} \wedge \underset{s_3}{(u_1 \vee \neg u_2 \vee e_2)} \wedge \underset{s_4}{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2 \qquad S' = \{C_1 \wedge C_2 \wedge C_3\}$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2 \qquad S' = \{C_1 \wedge C_2 \wedge C_3\}$$

$$\mathscr{A} = \mathscr{A} \wedge (s_4)$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2 \qquad S' = \{C_1 \wedge C_2 \wedge C_3\}$$

$$\mathscr{A} = \mathscr{A} \wedge (s_4)$$

$$\tau = \neg u_1 \wedge \neg u_2 \wedge s_1 \wedge s_2 \wedge s_4$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2 \qquad S' = \{C_1 \wedge C_2 \wedge C_3\}$$

$$\mathscr{A} = \mathscr{A} \wedge (s_4)$$

$$\tau = \neg u_1 \wedge \neg u_2 \wedge s_1 \wedge s_2 \wedge s_4 \qquad (e_1) \wedge (e_2) \wedge (\neg e_1 \vee \neg e_2)$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2 \qquad S' = \{C_1 \wedge C_2 \wedge C_3\}$$

$$\mathscr{A} = \mathscr{A} \wedge (s_4)$$

$$\tau = \neg u_1 \wedge \neg u_2 \wedge s_1 \wedge s_2 \wedge s_4$$

$$(e_1) \wedge (e_2) \wedge (\neg e_1 \vee \neg e_2)$$

$$(\neg e_1)$$

CEGAR with Clause Selection

Janota, Marques-Silva 2015 Rabe, Tentrup 2015

```
def CEGAR_Sel():
  𝒜 = {(¬s_C ∨ ¬ℓ) | C ∈ φ, ℓ ∈ C, var(ℓ) ∈ U}
  while True:
    abs_sat, τ = SAT(𝒜)
    if not abs_sat:
      return True
    S = {C ∈ φ | τ(s_C) = ⊤ }
    φ' = ⋀_{C∈S} ⋁_{ℓ∈C, var(ℓ)∈E} ℓ
    mat_sat, σ = SAT(φ')
    if not mat_sat:
      return False
    S' = {C ∈ φ | σ ⊨ C}
    # Refine
    𝒜 = 𝒜 ∧ ⋁_{C∈φ\S'} s_C
```

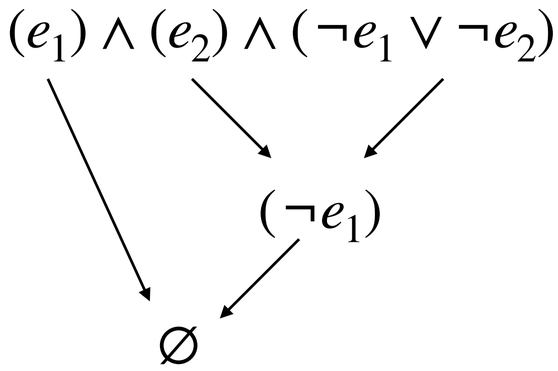
$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$\underbrace{(u_1 \vee e_1)}_{s_1} \wedge \underbrace{(u_2 \vee e_2)}_{s_2} \wedge \underbrace{(u_1 \vee \neg u_2 \vee e_2)}_{s_3} \wedge \underbrace{(u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)}_{s_4}$$

$$\tau = \neg u_1 \wedge u_2 \wedge s_1 \wedge s_3 \qquad \sigma = e_1 \wedge e_2 \qquad S' = \{C_1 \wedge C_2 \wedge C_3\}$$

$$\mathscr{A} = \mathscr{A} \wedge (s_4)$$

$$\tau = \neg u_1 \wedge \neg u_2 \wedge s_1 \wedge s_2 \wedge s_4$$



More on Clause Selection

More on Clause Selection

can be generalized from PCNF

More on Clause Selection

can be generalized from PCNF

Janota 2018

More on Clause Selection

can be generalized from PCNF

Janota 2018

Tentrup 2016

More on Clause Selection

can be generalized from PCNF

prenex QBF

Janota 2018

Tentrup 2016

More on Clause Selection

can be generalized from PCNF

prenex QBF

Janota 2018

Tentrup 2016

work on the underlying proof system ∀Red+Res

More on Clause Selection

can be generalized from PCNF

prenex QBF

Janota 2018

Tentrup 2016

work on the underlying proof system ∀Red+Res

Tentrup 2017

More on Clause Selection

can be generalized from PCNF

prenex QBF

Janota 2018

Tentrup 2016

work on the underlying proof system ∀Red+Res

Tentrup 2017

∀Red+Res equivalent to level-ordered Q-resolution

More on Clause Selection

can be generalized from PCNF

prenex QBF

Janota 2018

Tentrup 2016

non-prenex QBF

work on the underlying proof system ∀Red+Res

Tentrup 2017

∀Red+Res equivalent to
level-ordered Q-resolution

4. Incremental Determinization

for ∀∃-2QBF

4. Incremental Determinization

Unique Strategy Functions

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i$$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$\neg C_1 \rightarrow \neg e$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e$$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e \quad \ldots$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e \quad \ldots \quad \neg C_m \rightarrow e$$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e \quad \ldots \quad \neg C_m \rightarrow e \quad e$ **deterministic** if $\bigwedge_{i=1}^{m} C_i$ UNSAT

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e \quad \ldots \quad \neg C_m \rightarrow e$ e **deterministic** if $\bigwedge_{i=1}^{m} C_i$ UNSAT

$\neg C_i \rightarrow e$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e \quad \ldots \quad \neg C_m \rightarrow e \quad e$ **deterministic** if $\bigwedge_{i=1}^{m} C_i$ UNSAT

$$\neg C_i \rightarrow e \qquad \neg C_j \rightarrow \neg e$$

Unique Strategy Functions

$$\forall u_1, \ldots, \forall u_n \exists e_1, \ldots, \exists e_n \; . \bigwedge_{i=1}^{n} (u_i \vee \neg e_i) \wedge (\neg u_i \vee e_i)$$

$f_{e_i} := u_i$ in the unique existential winning strategy

$$\neg u_i \rightarrow \neg e_i \qquad u_i \rightarrow e_i$$

$\neg C_1 \rightarrow \neg e \quad \neg C_2 \rightarrow e \quad \ldots \quad \neg C_m \rightarrow e$ e **deterministic** if $\bigwedge_{i=1}^{m} C_i$ UNSAT

$\neg C_i \rightarrow e \quad \neg C_j \rightarrow \neg e$ e **conflicted** if $\neg C_i \wedge \neg C_j$ SAT

Lifting CDCL to Strategies

Lifting CDCL to Strategies

Lifting CDCL to Strategies

	CDCL

Lifting CDCL to Strategies

	CDCL	Incremental Determinization

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail		

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	

Lifting CDCL to Strategies

	CDCL	**Incremental Determinization**
trail	variable assignment	partial existential strategy

Lifting CDCL to Strategies

	CDCL	**Incremental Determinization**
trail	variable assignment	partial existential strategy
propagation		

Lifting CDCL to Strategies

	CDCL	**Incremental Determinization**
trail	variable assignment	partial existential strategy
propagation	unit clause	

Lifting CDCL to Strategies

	CDCL	**Incremental Determinization**
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function

Lifting CDCL to Strategies

	CDCL	**Incremental Determinization**
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision		

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict		

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict	unit clauses (y) (¬y)	

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict	unit clauses (y) (¬y)	conflicted existential variable

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict	unit clauses (y) (¬y)	conflicted existential variable
learning		

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict	unit clauses (y) (¬y)	conflicted existential variable
learning	clause	

Lifting CDCL to Strategies

	CDCL	Incremental Determinization
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict	unit clauses (y) (¬y)	conflicted existential variable
learning	clause	clause

Lifting CDCL to Strategies

Rabe and Seshia 2016

	CDCL	**Incremental Determinization**
trail	variable assignment	partial existential strategy
propagation	unit clause	unique strategy function
decision	literal	clauses ensuring determinicity
conflict	unit clauses (y) (¬y)	conflicted existential variable
learning	clause	clause

Propagation and Conflicts

Propagation and Conflicts

```
def propagate(∀U∃E, D):
```

Propagation and Conflicts

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

Propagation and Conflicts

clause $C \rightarrow e$ or $C \rightarrow \neg e$
and $var(C) \subseteq D \cup U$

$\forall u \exists e_1 \exists e_2 \exists e_3$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

$$I_2 = \{e_2, e_3\}$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

$$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

e_2 deterministic when $e_1 := \neg u_1$

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

$$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

e_2 deterministic when $e_1 := \neg u_1$

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

$$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$$

$$I_3 = \{e_3\}$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

e_2 deterministic when $e_1 := \neg u_1$

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

$$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$$

$$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$$

Propagation and Conflicts

clause $C \to e$ or $C \to \neg e$
and $var(C) \subseteq D \cup U$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D)≠∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

e_2 deterministic when $e_1 := \neg u_1$

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge$$

$$(e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$$

$$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$$

$$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$$

e_3 conflicted under $\neg u \wedge \neg e_2 \wedge e_1$

Decisions

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

$$(u_1)$$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

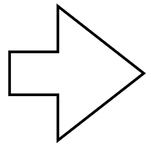
(u_1) SAT

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT

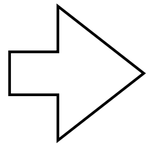


Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT



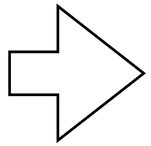
e_1 **not** deterministic

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT



e_1 **not** deterministic

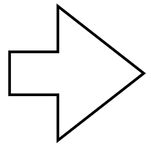
$$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT



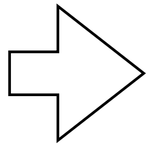
e_1 **not** deterministic

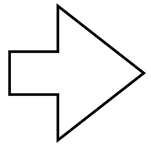
$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT  e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT 

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT ⇨ e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT ⇨ e_2 **not** deterministic

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT ⇨ e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT ⇨ e_2 **not** deterministic

make e_2 deterministic:

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT ⇨ e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT ⇨ e_2 **not** deterministic

make e_2 deterministic: if $\neg e_2$ is not implied, set $e_2 := \top$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT ⇨ e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT ⇨ e_2 **not** deterministic

make e_2 deterministic: if $\neg e_2$ is not implied, set $e_2 := \top$

$$u_2 \rightarrow e_2$$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT ⇨ e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT ⇨ e_2 **not** deterministic

make e_2 deterministic: if $\neg e_2$ is not implied, set $e_2 := \top$

$u_2 \rightarrow e_2$ general case: $\left(\bigwedge_{C : \neg e \in C} C \setminus \{\neg e\} \right) \rightarrow e$

Decisions

$$\forall u_1 \forall u_2 \exists e_1 \exists e_2$$

$$(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2) \wedge (u_1 \vee \neg u_2 \vee e_2) \wedge (u_1 \vee u_2 \vee \neg e_1 \vee \neg e_2)$$

(u_1) SAT ⇨ e_1 **not** deterministic

$(u_1 \vee \neg u_2) \wedge (u_1 \wedge u_2)$ SAT ⇨ e_2 **not** deterministic

make e_2 deterministic: if $\neg e_2$ is not implied, set $e_2 := \top$

$u_2 \rightarrow e_2$

general case: $\left(\bigwedge_{C : \neg e \in C} C \setminus \{\neg e\} \right) \rightarrow e$

need not be CNF

Conflict Analysis

Conflict Analysis

$$\tau : U \to \{ \bot , \top \}$$

Conflict Analysis

$\tau : U \to \{ \bot , \top \}$

partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$ partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$

partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$

partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$

partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$

$\tau \cup \sigma \models \psi_D$

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$

partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$

$\tau \cup \sigma \models \psi_D$

$\tau \cup \sigma \models \neg\varphi$

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$ partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$ $\tau \cup \sigma \models \psi_D$ $\tau \cup \sigma \models \neg\varphi$

perform CDCL conflict analysis:

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$ partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$ $\tau \cup \sigma \models \psi_D$ $\tau \cup \sigma \models \neg\varphi$

perform CDCL conflict analysis: deterministic ~ propagated

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$ partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$ $\tau \cup \sigma \models \psi_D$ $\tau \cup \sigma \models \neg\varphi$

perform CDCL conflict analysis: deterministic ~ propagated

decisions ~ decisions

Conflict Analysis

clauses from φ and decision constraints

$\tau : U \to \{ \bot , \top \}$ partial strategy $\psi_D := \bigwedge_{y \in D} \psi_y$

unique $\sigma : D \to \{ \top , \bot \}$ $\tau \cup \sigma \models \psi_D$ $\tau \cup \sigma \models \neg\varphi$

perform CDCL conflict analysis:

deterministic ~ propagated

decisions ~ decisions

universal variables get decision level 0

Conflict Analysis

$\forall u \exists e_1 \exists e_2 \exists e_3$ $(\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$

Conflict Analysis

$\forall u \exists e_1 \exists e_2 \exists e_3$ $(\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$

decision on e_1

Conflict Analysis

$$\forall u \, \exists e_1 \, \exists e_2 \, \exists e_3 \;\; (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

decision on e_1

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

decision on e_1 Decision Level 1

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1

Decision Level 1

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1

Decision Level 1

$I_1 = \{e_1, e_2\}$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1

Decision Level 1

$I_1 = \{e_1, e_2\}$ $D_1 = \emptyset$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\}$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}$ $D_1 = \emptyset$

$I_2 = \{e_2, e_3\}$ $D_2 = \{e_1\}$

$I_3 = \{e_3\}$ $D_2 = \{e_1, e_2\}$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$(u \vee e_2 \vee \neg e_1)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$(u \vee e_2 \vee \neg e_1)$$

```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

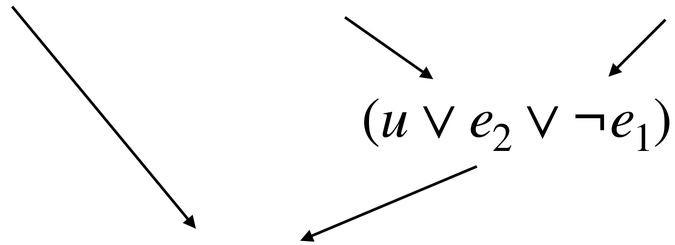
$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$




```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

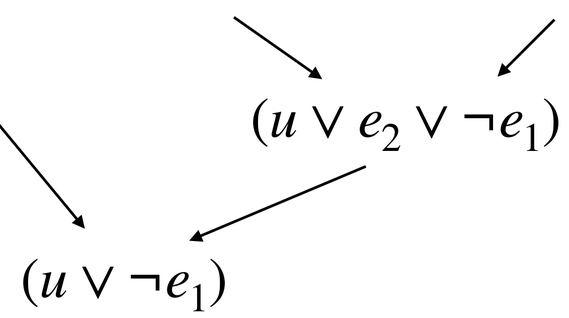
$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$




```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

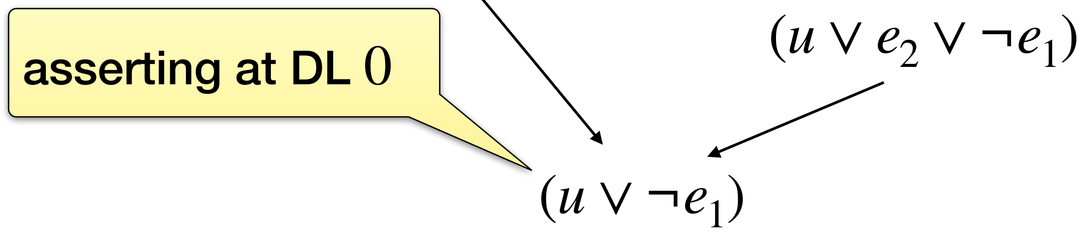
$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Conflict Analysis

$$\forall u \exists e_1 \exists e_2 \exists e_3 \ (u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge (u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$




```
def propagate(∀U∃E, D):
  I = [e for e in E\D if e is_unique_consequence(e,D)]
  while I∩(E\D) ≠ ∅:
    e = I.pop()
    if is_deterministic(e):
      if is_conflicted(e):
        return e, get_conflicting_assignment(e)
      else:
        D.add(e)
```

decision on e_1 Decision Level 1

$I_1 = \{e_1, e_2\}\ D_1 = \emptyset$

$I_2 = \{e_2, e_3\}\ D_2 = \{e_1\}$

$I_3 = \{e_3\} \quad D_2 = \{e_1, e_2\}$

e_3 conflicted under $\neg u \wedge e_1 \wedge \neg e_2 \wedge e_3$

Incremental Determinization

Incremental Determinization

```
def Incremental_Determinization():
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1\}$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1\} \qquad D_1 = \emptyset$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1\} \qquad D_1 = \emptyset \qquad \psi_{e_1} := \neg e_1$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1\} \qquad D_1 = \emptyset \qquad \psi_{e_1} := \neg e_1$$

$$I_2 = \{e_2\}$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$$I_1 = \{e_1\} \quad D_1 = \emptyset \quad \psi_{e_1} := \neg e_1$$

$$I_2 = \{e_2\} \quad D_2 = \{e_1\}$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$I_1 = \{e_1\}$ $D_1 = \emptyset$ $\psi_{e_1} := \neg e_1$

$I_2 = \{e_2\}$ $D_2 = \{e_1\}$ $(e_1) \wedge (u) \wedge \psi_{e_1}$ UNSAT

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$I_1 = \{e_1\}$ $D_1 = \emptyset$ $\psi_{e_1} := \neg e_1$

$I_2 = \{e_2\}$ $D_2 = \{e_1\}$ $(e_1) \wedge (u) \wedge \psi_{e_1}$ UNSAT

e_2 conflicted under $\neg u \wedge \neg e_1$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$I_1 = \{e_1\}$ $D_1 = \emptyset$ $\psi_{e_1} := \neg e_1$

$I_2 = \{e_2\}$ $D_2 = \{e_1\}$ $(e_1) \wedge (u) \wedge \psi_{e_1}$ UNSAT

e_2 conflicted under $\neg u \wedge \neg e_1$

$$(u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2)$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

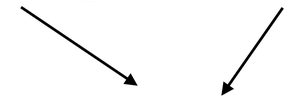
$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$I_1 = \{e_1\}$ $D_1 = \emptyset$ $\psi_{e_1} := \neg e_1$

$I_2 = \{e_2\}$ $D_2 = \{e_1\}$ $(e_1) \wedge (u) \wedge \psi_{e_1}$ UNSAT

e_2 conflicted under $\neg u \wedge \neg e_1$

$$(u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2)$$



$$(u \vee e_1)$$

Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

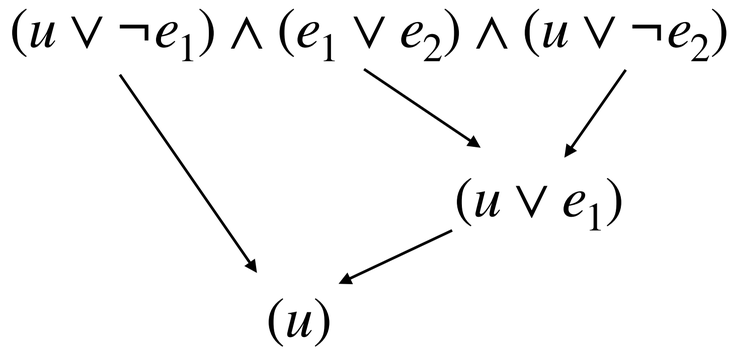
$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$I_1 = \{e_1\}$ $D_1 = \emptyset$ $\psi_{e_1} := \neg e_1$

$I_2 = \{e_2\}$ $D_2 = \{e_1\}$ $(e_1) \wedge (u) \wedge \psi_{e_1}$ UNSAT

e_2 conflicted under $\neg u \wedge \neg e_1$



Incremental Determinization

```
def Incremental_Determinization():
  while True:
    conflict = propagate()
    if conflict is not None:
      clause, bt_level = analyze(conflict)
      if is_universal(clause):
        return False
      attach(clause)
      backtrack(bt_level)
    elif D = E:
      return True
    else:
      decide()
```

$$\forall u \exists e_1 \exists e_2 \exists e_3$$

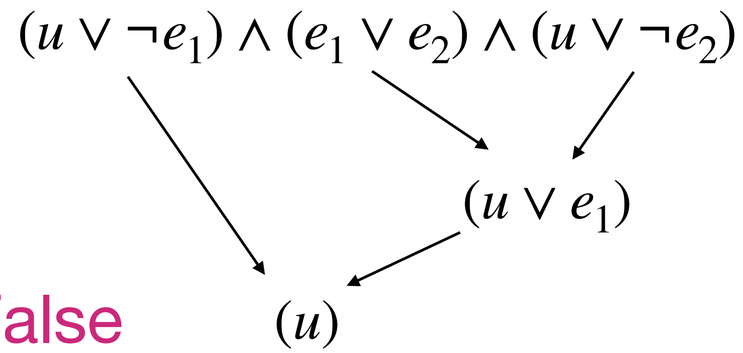
$$(u \vee \neg e_1) \wedge (\neg u \vee \neg e_1) \wedge (e_1 \vee e_2) \wedge (u \vee \neg e_2) \wedge$$

$$(u \vee e_2 \vee e_3) \wedge (\neg e_1 \vee \neg e_3)$$

$I_1 = \{e_1\}$ $D_1 = \emptyset$ $\psi_{e_1} := \neg e_1$

$I_2 = \{e_2\}$ $D_2 = \{e_1\}$ $(e_1) \wedge (u) \wedge \psi_{e_1}$ UNSAT

e_2 conflicted under $\neg u \wedge \neg e_1$



Input 2QBF False

More on Incremental Determinization

More on Incremental Determinization

can be used for Boolean Functional Synthesis

More on Incremental Determinization

can be used for Boolean Functional Synthesis

Rabe 2019

More on Incremental Determinization

can be used for Boolean Functional Synthesis

Rabe 2019

formalization as a proof system and combination with CEGAR

More on Incremental Determinization

can be used for Boolean Functional Synthesis

Rabe 2019

formalization as a proof system and combination with CEGAR

Rabe, Tentrup, Rasmussen, Seshia 2018

Further Topics

Further Topics

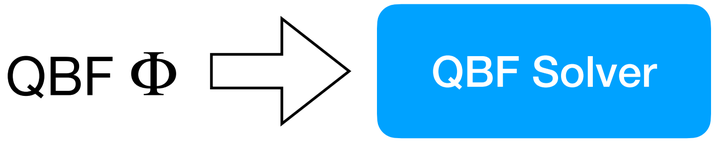
QBF Solver

Further Topics

QBF Φ

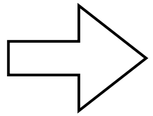
QBF Solver

Further Topics



Further Topics

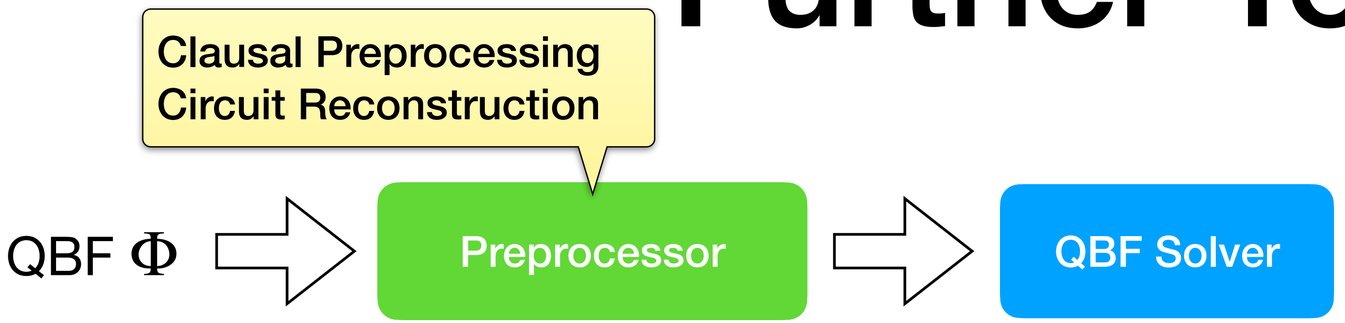
QBF Φ



QBF Solver

Further Topics

Further Topics

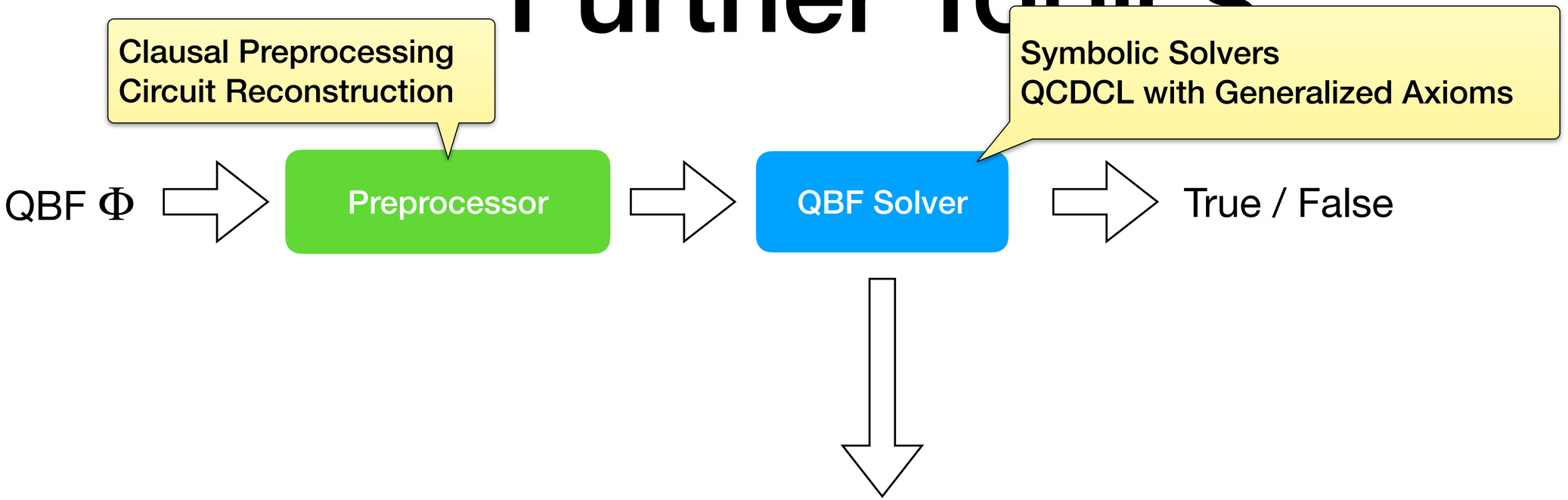


Further Topics

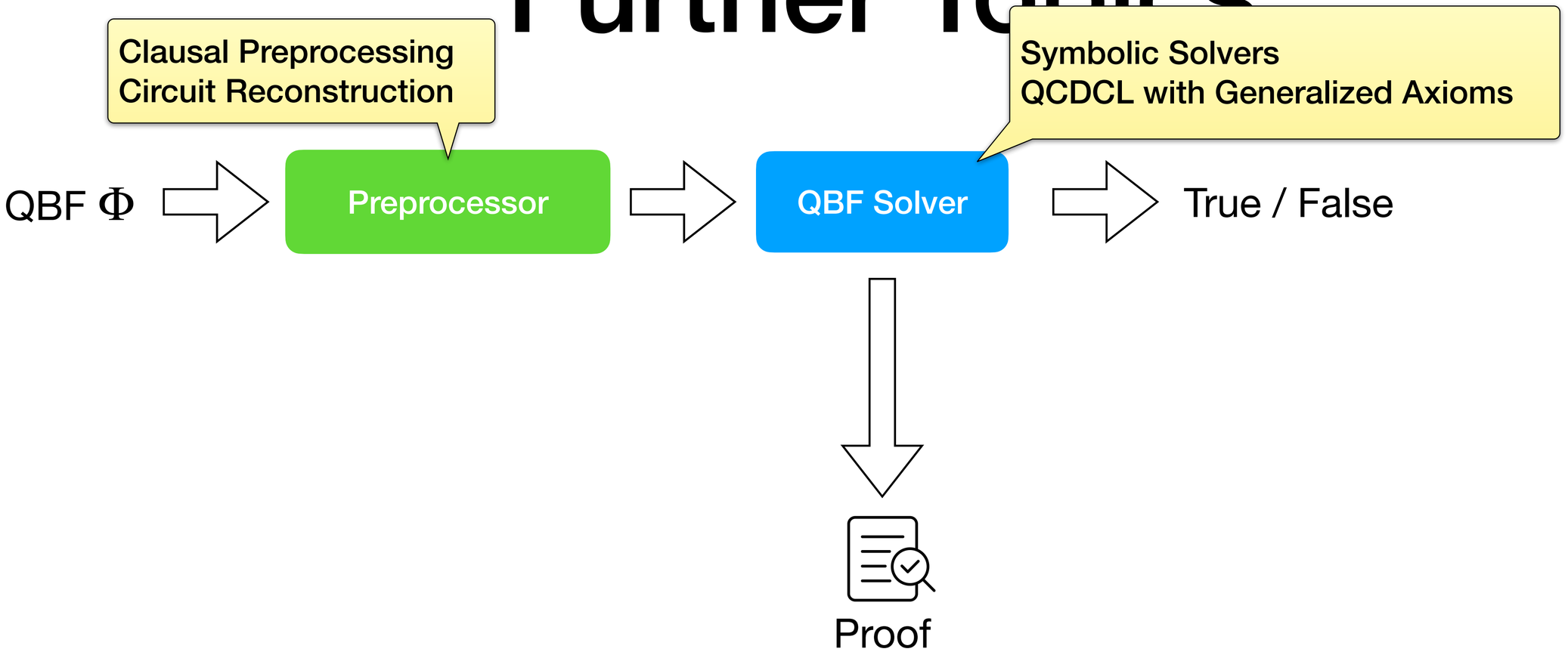
Further Topics

Further Topics

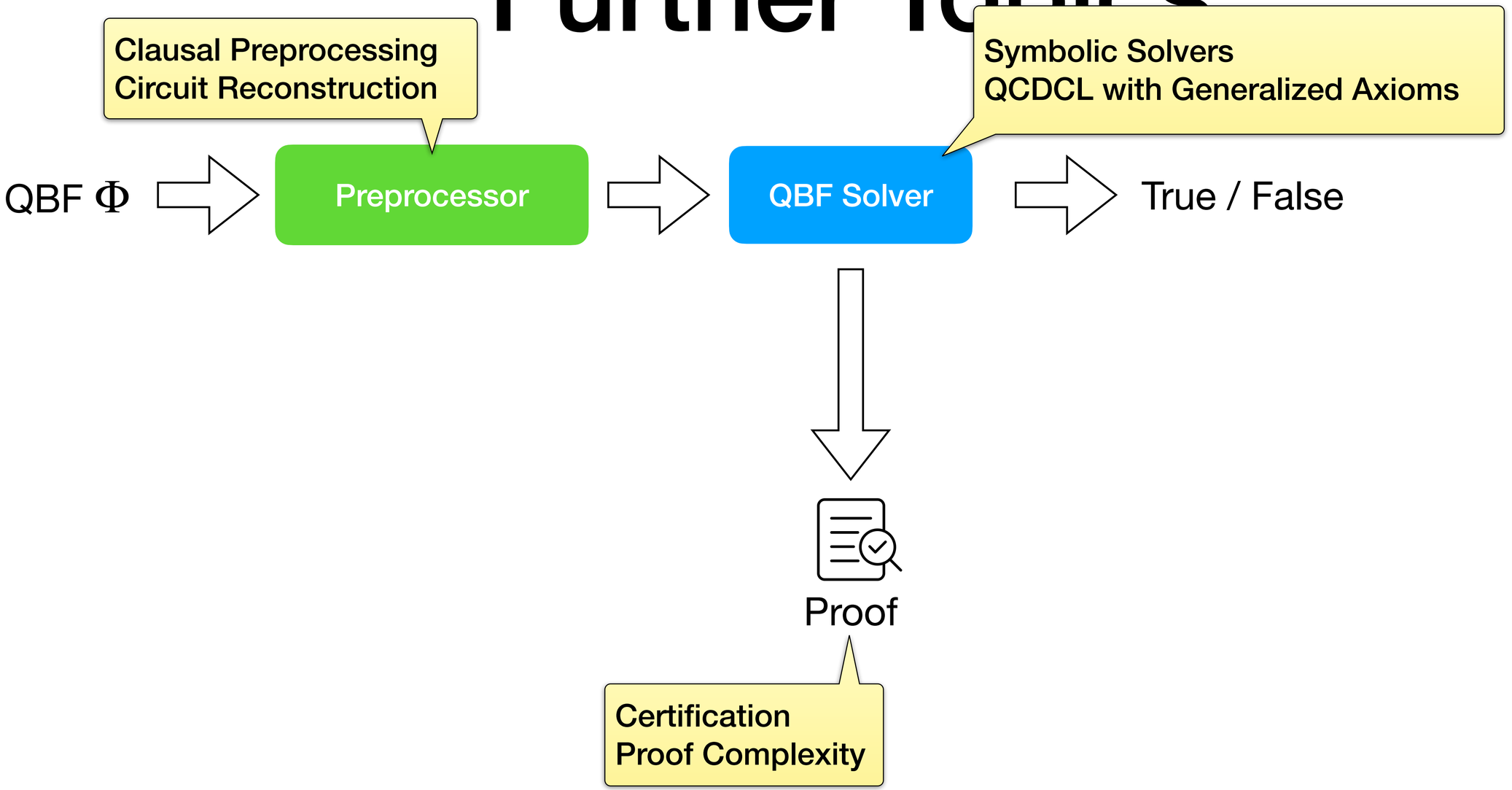
Further Topics



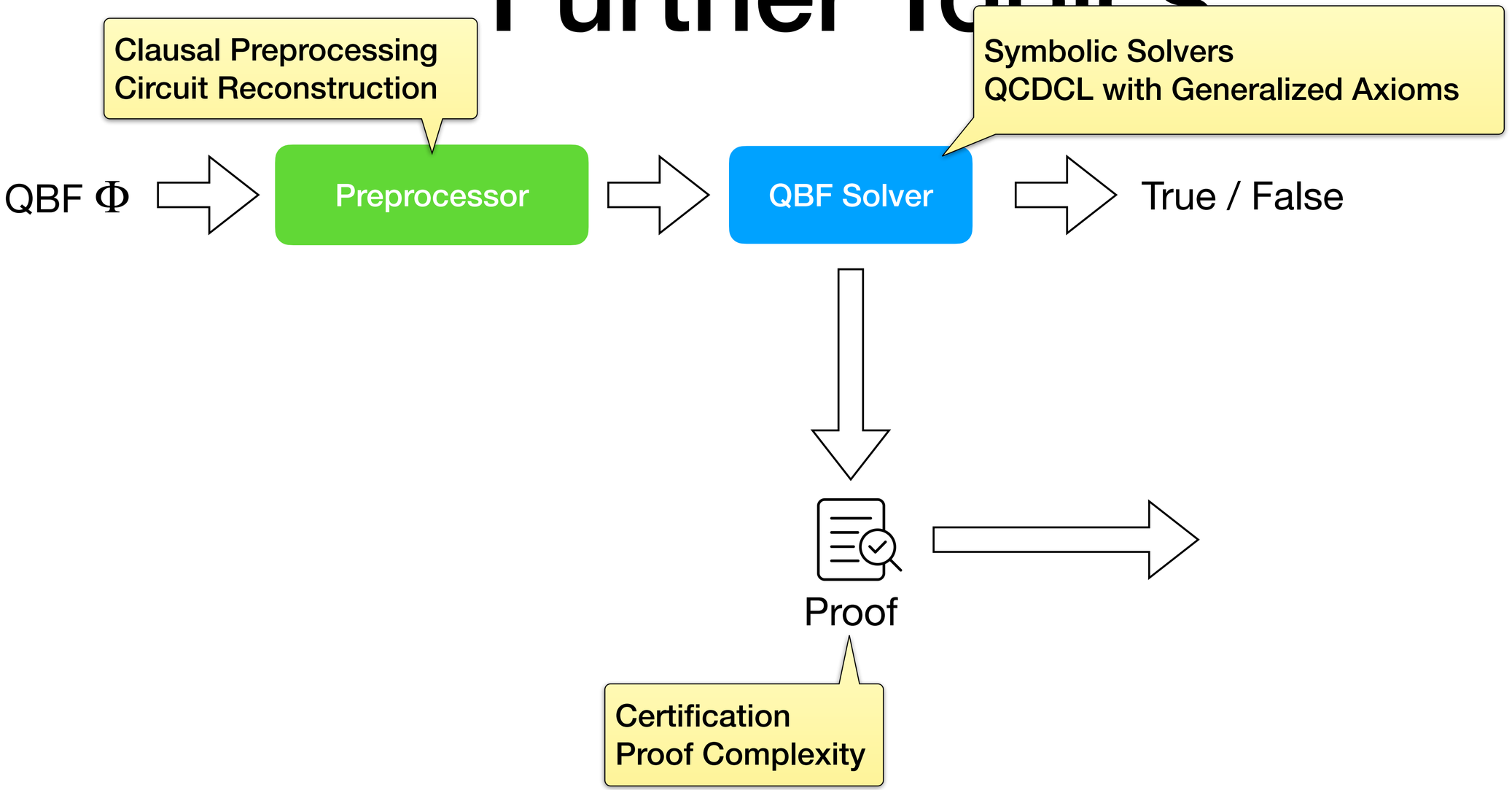
Further Topics



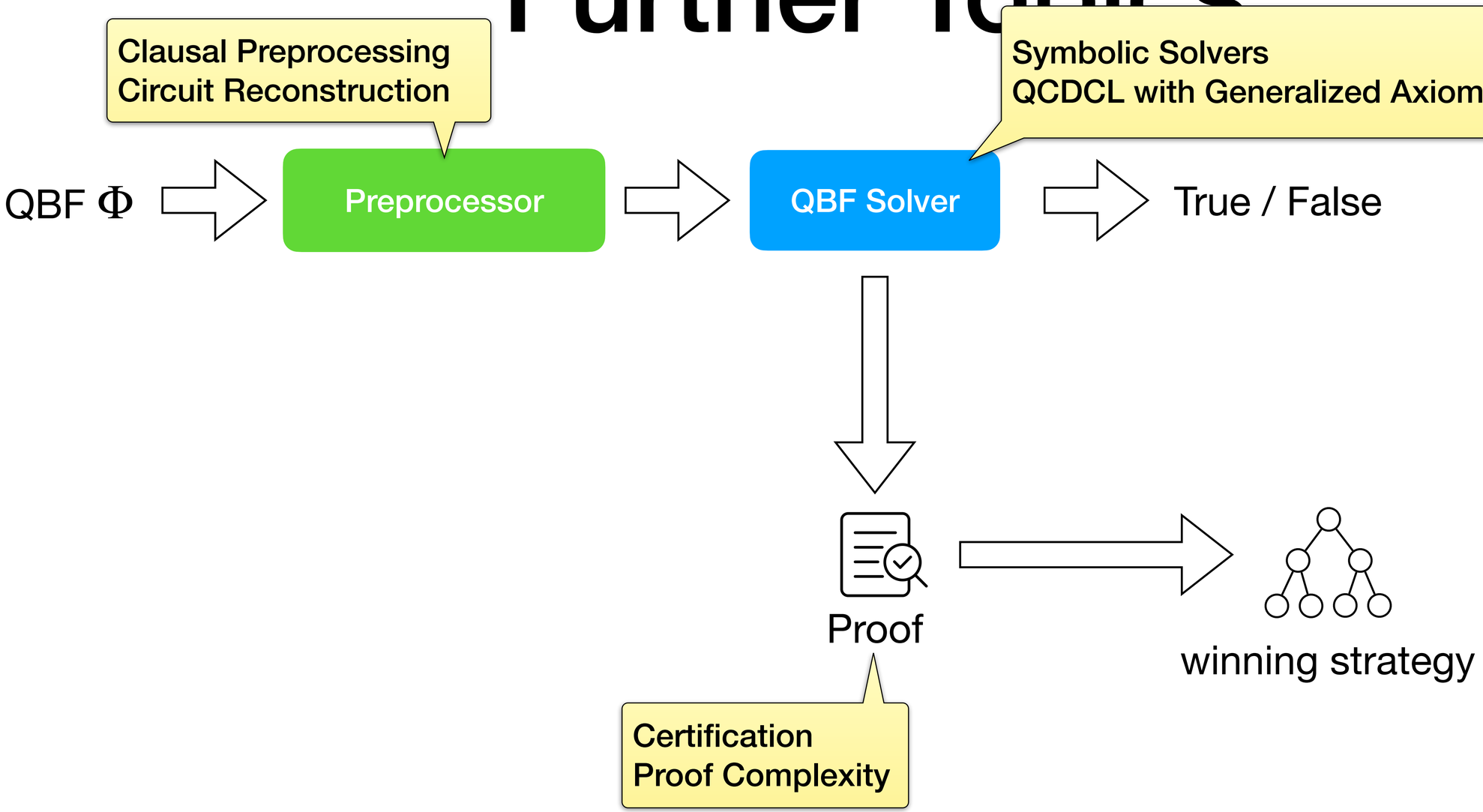
Further Topics



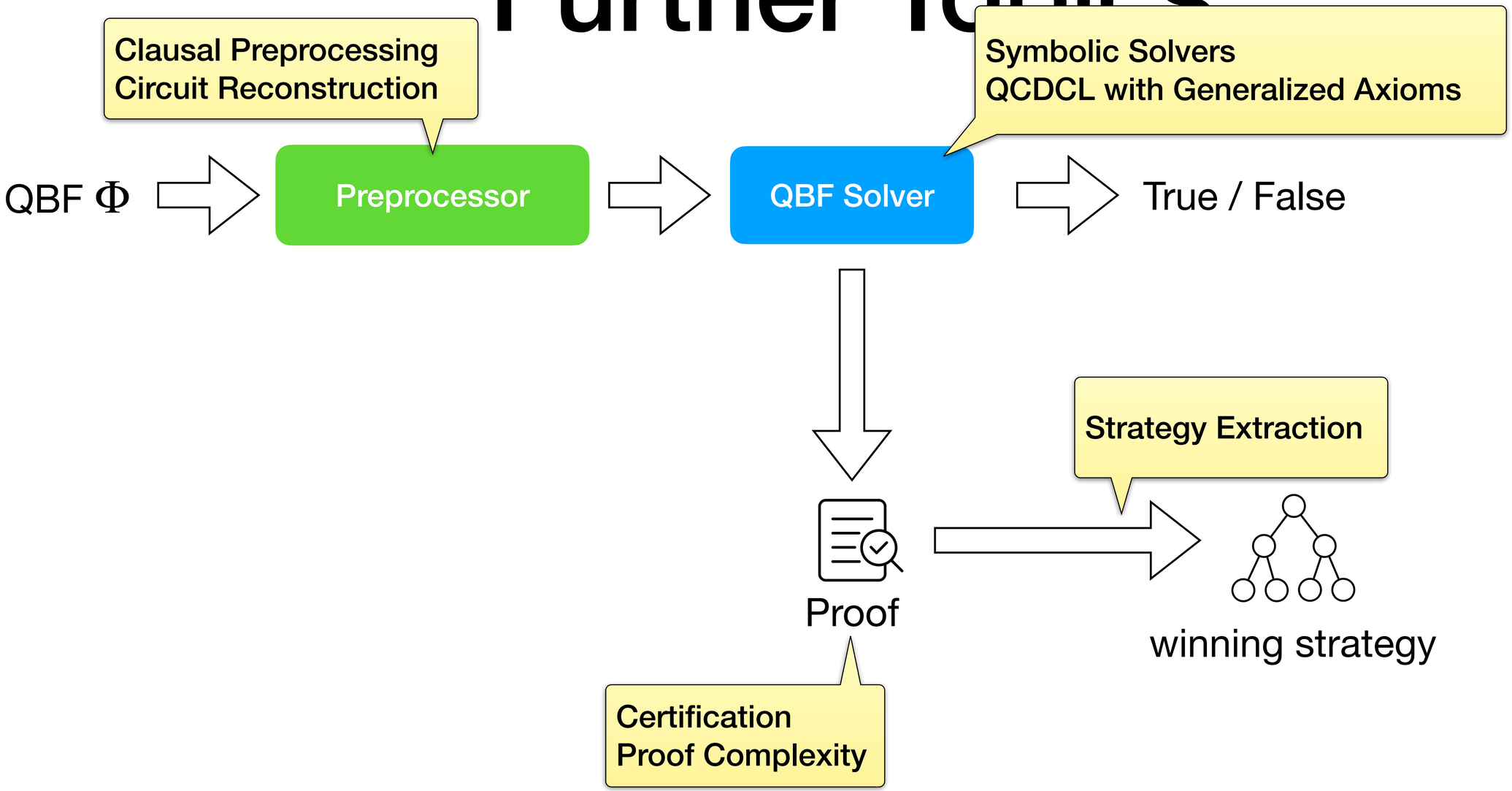
Further Topics



Further Topics



Further Topics



Conclusion

Let a thousand solvers run, let a hundred proof systems contend.

Conclusion

Let a thousand solvers run, let a hundred proof systems contend.

There's **room for experimentation** and tinkering with QBF solver architectures.

Conclusion

Let a thousand solvers run, let a hundred proof systems contend.

There's **room for experimentation** and tinkering with QBF solver architectures.

A **"killer application"** for QBF solving is still missing.

Conclusion

Let a thousand solvers run, let a hundred proof systems contend.

There's **room for experimentation** and tinkering with QBF solver architectures.

A **"killer application"** for QBF solving is still missing.

There's **much to do** both in solver development and theory.