Propositional Theorem Proving

Outline

I. The resolution algorithm

II. Horn clauses

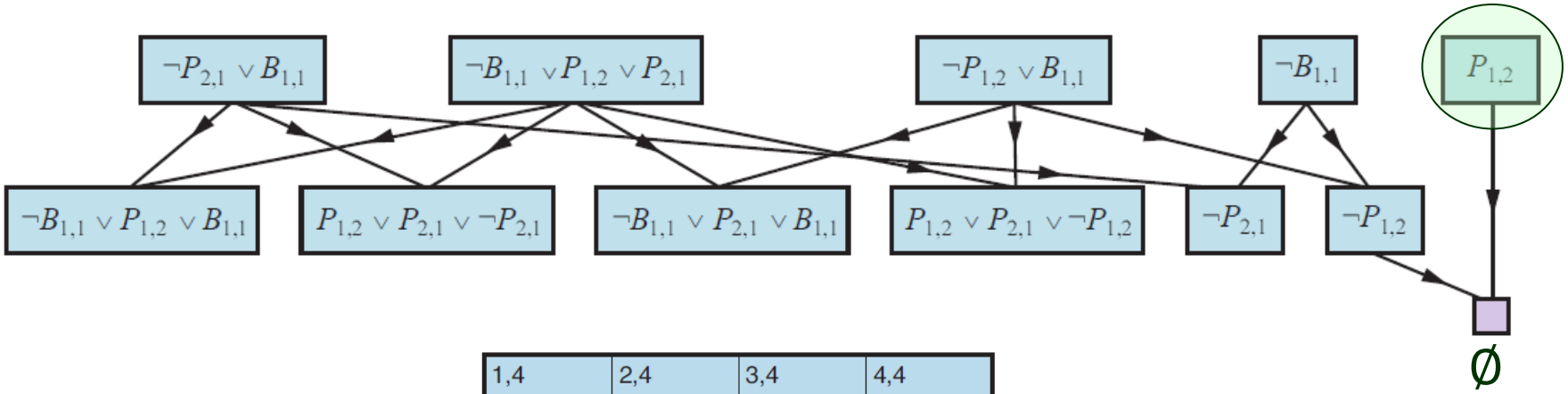
III. Forward and backward chaining

IV. Effective propositional model checking

* Figures are from the textbook site unless a source is specifically cited.

I. Proving $\neg P_{1,2}$ in the Wumpus World

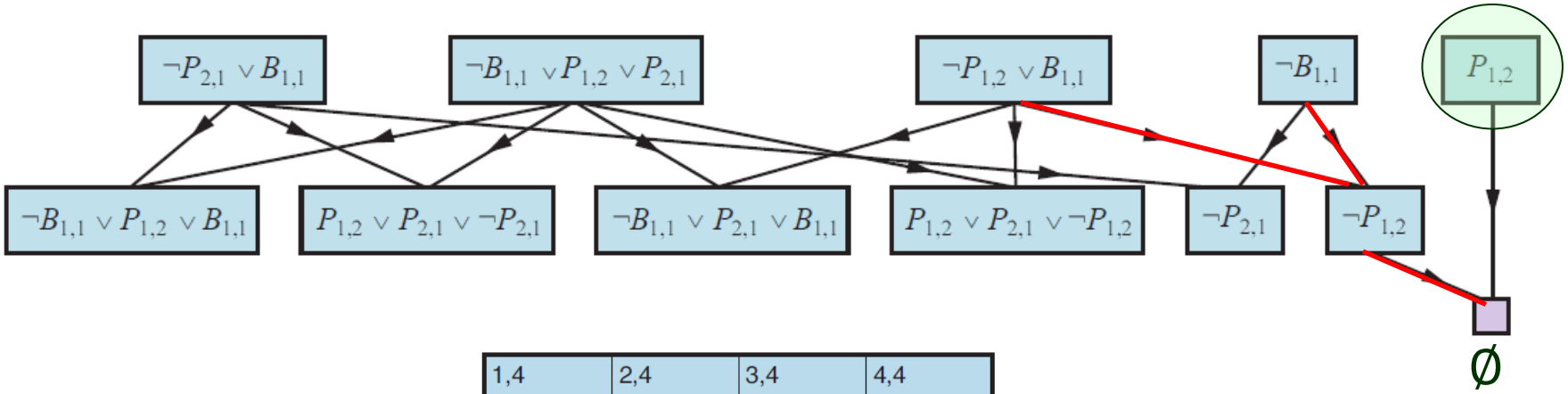
$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ transformed into CNF: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$



1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

I. Proving $\neg P_{1,2}$ in the Wumpus World

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ transformed into CNF: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$



1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

Resolution Algorithm

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\,\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** $clauses$ **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if $resolvents$ contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false* // no new clauses can be added.
    $clauses \leftarrow clauses \cup new$

The process ends in one of two situations below:

- No new clauses can be added, in which case *KB* does not entail α;
- Two clauses resolve to yield the empty clause, in which case *KB* entails α.

Completeness of Resolution

Given a set of clauses S, its *resolution closure* $RC(S)$ includes all the clauses in S as well as all the resolvents from repeated applications of the resolution rule.

Completeness of Resolution

Given a set of clauses S, its *resolution closure* $RC(S)$ includes all the clauses in S as well as all the resolvents from repeated applications of the resolution rule.

$RC(S)$ is finite because only 3^n distinct clauses can be constructed out of n propositional symbols appearing in S.

// each symbol appears in a clause as P or $\neg P$, or does not appear.

Completeness of Resolution

Given a set of clauses S, its *resolution closure* $RC(S)$ includes all the clauses in S as well as all the resolvents from repeated applications of the resolution rule.

> $RC(S)$ is finite because only 3^n distinct clauses can be constructed out of n propositional symbols appearing in S.
>
> // each symbol appears in a clause as P or $\neg P$, or does not appear.

Ground Resolution Theorem: If S is unsatisfiable, then $RC(S)$ contains the empty clause $\emptyset$.

Completeness of Resolution

Given a set of clauses S, its *resolution closure* $RC(S)$ includes all the clauses in S as well as all the resolvents from repeated applications of the resolution rule.

$RC(S)$ is finite because only 3^n distinct clauses can be constructed out of n propositional symbols appearing in S.

// each symbol appears in a clause as P or $\neg P$, or does not appear.

Ground Resolution Theorem: If S is unsatisfiable, then $RC(S)$ contains the empty clause $\emptyset$.

$RC(S)$ does not contain the empty clause $\emptyset \Rightarrow S$ is satisfiable.

Completeness of Resolution

Given a set of clauses S, its *resolution closure* $RC(S)$ includes all the clauses in S as well as all the resolvents from repeated applications of the resolution rule.

$RC(S)$ is finite because only 3^n distinct clauses can be constructed out of n propositional symbols appearing in S.

// each symbol appears in a clause as P or $\neg P$, or does not appear.

Ground Resolution Theorem: If S is unsatisfiable, then $RC(S)$ contains the empty clause $\emptyset$.

$RC(S)$ does not contain the empty clause $\emptyset \Rightarrow S$ is satisfiable.

We can construct a proof by explicitly generating an assignment for S if $\emptyset \notin RC(S)$.

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

```
¬ rain ∨ ¬ outside ∨ wet
```

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

♣ *Fact* (1 positive literal and 0 negative literal)

$$P_{1,2}$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

♣ *Fact* (1 positive literal and 0 negative literal)

$$P_{1,2} \quad \equiv \quad \textit{true} \Rightarrow P_{1,2}$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

♣ *Fact* (1 positive literal and 0 negative literal)

$$P_{1,2} \quad \equiv \quad \textit{true} \Rightarrow P_{1,2}$$

- *Goal clause* (0 positive literal and ≥ 1 negative literal)

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

♣ *Fact* (1 positive literal and 0 negative literal)

$$P_{1,2} \quad \equiv \quad \textit{true} \Rightarrow P_{1,2}$$

- *Goal clause* (0 positive literal and ≥ 1 negative literal)

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow \textit{false}$$

II. Horn Clauses

A clause is called a *Horn clause* if it contains ≤ 1 positive literal.

P: positive literal $\neg P$: negative literal

- *Definite clause* (1 positive literal and ≥ 0 negative literal)

$$\neg P \vee \neg Q \vee R \quad \equiv \quad P \wedge Q \Rightarrow R$$

$$\neg \texttt{rain} \vee \neg \texttt{outside} \vee \texttt{wet} \quad \equiv \quad \texttt{rain} \wedge \texttt{outside} \Rightarrow \texttt{wet}$$

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

♣ *Fact* (1 positive literal and 0 negative literal)

$$P_{1,2} \quad \equiv \quad \textit{true} \Rightarrow P_{1,2}$$

- *Goal clause* (0 positive literal and ≥ 1 negative literal)

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow \textit{false}$$

// We are trying to prove $P_1 \wedge P_2 \wedge \cdots \wedge P_k$.

Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

$$\Downarrow$$

$$Q \Leftarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

$$\Downarrow$$

$$Q \Leftarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

$$\Downarrow$$

Q :- P1, P2, …, Pk.

(Prolog programming language)

Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

$$\Downarrow$$

$$Q \Leftarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

$$\Downarrow$$

Q :- P1, P2, …, Pk.

(Prolog programming language)

- Horn clauses are *closed* under resolution, i.e., the resolvent of two Horn clauses is still a Horn clause.

Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

$$\Downarrow$$

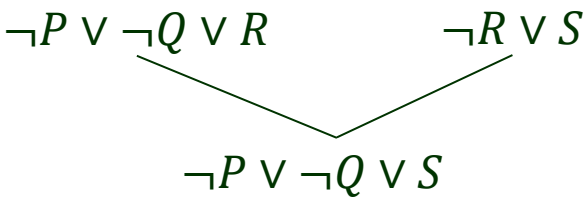
$$Q \Leftarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

$$\Downarrow$$

Q :- P1, P2, …, Pk.

(Prolog programming language)

- Horn clauses are *closed* under resolution, i.e., the resolvent of two Horn clauses is still a Horn clause.



Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

$$\Downarrow$$

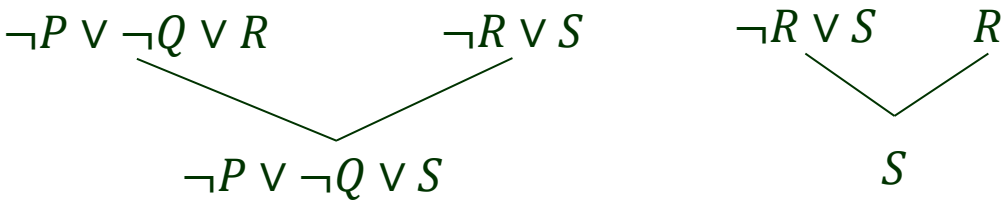
$$Q \Leftarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

$$\Downarrow$$

Q :- P1, P2, …, Pk.

(Prolog programming language)

- Horn clauses are *closed* under resolution, i.e., the resolvent of two Horn clauses is still a Horn clause.



Why Horn Clauses?

Horn clauses are the basis of logic programming, and play an important role in automated theorem proving.

- Every definite clause can be written as an implication.

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_k \vee Q \quad \equiv \quad (P_1 \wedge P_2 \wedge \cdots \wedge P_k) \Rightarrow Q$$

$$\Downarrow$$

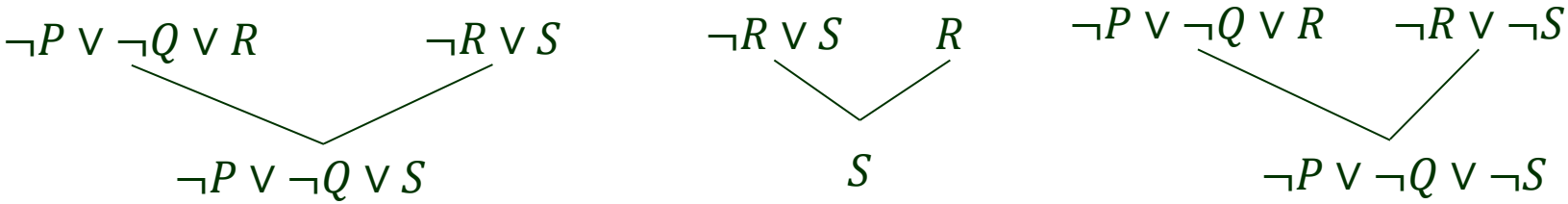
$$Q \Leftarrow (P_1 \wedge P_2 \wedge \cdots \wedge P_k)$$

$$\Downarrow$$

Q :- P1, P2, …, Pk.

(Prolog programming language)

- Horn clauses are *closed* under resolution, i.e., the resolvent of two Horn clauses is still a Horn clause.



(cont'd)

- Inferences with Horn clauses are through forward- and backward-chaining algorithms.

Logic programming

(natural inference steps easy for humans to follow)

- Low computational complexity: deciding entailment with Horn clauses takes $O(n)$ time.

size of the KB = sum of sizes of clauses in the KB

III. Forward Chaining

Question $KB \models q$?

single proposition symbol

- Begins from positive literals (facts).
- If all the premises of an implications are known, then add its conclusion to KB (as a new fact).
- Continues until q is added or no further inferences can be made.

III. Forward Chaining

Question $KB \models q$?

single proposition symbol

- Begins from positive literals (facts).
- If all the premises of an implications are known, then add its conclusion to KB (as a new fact).
- Continues until q is added or no further inferences can be made.

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*
  inputs: KB, the knowledge base, a set of propositional definite clauses
    q, the query, a proposition symbol
  count ← a table, where *count*[c] is initially the number of symbols in clause c's premise
  inferred ← a table, where *inferred*[s] is initially *false* for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while *queue* is not empty **do**
    p ← POP(*queue*)
    if $p = q$ **then return** *true*
    if *inferred*[p] = *false* **then**
      inferred[p] ← *true*
      for each clause c in KB where p is in c.PREMISE **do**
        decrement *count*[c]
        if *count*[c] = 0 **then** add c.CONCLUSION to *queue*
  return *false*

Example of Forward Chaining

KB:

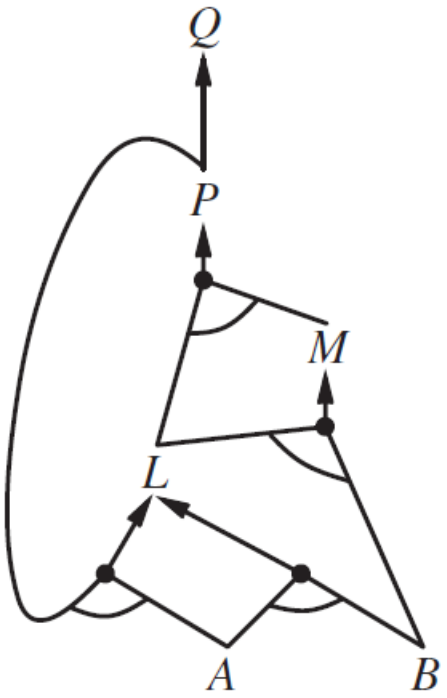
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Example of Forward Chaining

KB:

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

AND-OR graph representation

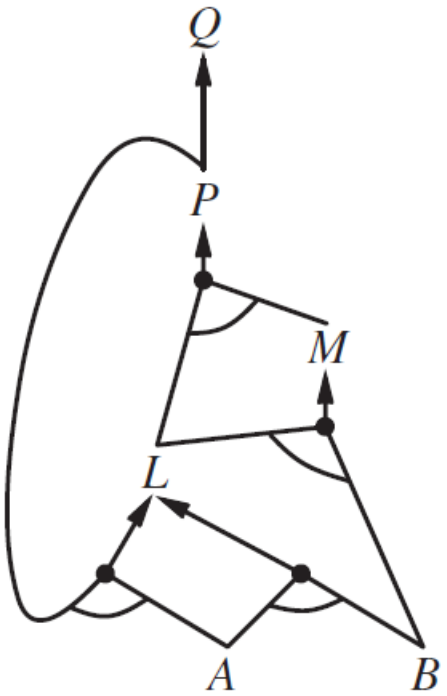


Example of Forward Chaining

KB:

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

AND-OR graph representation



Q. *KB* $\models Q$?

Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

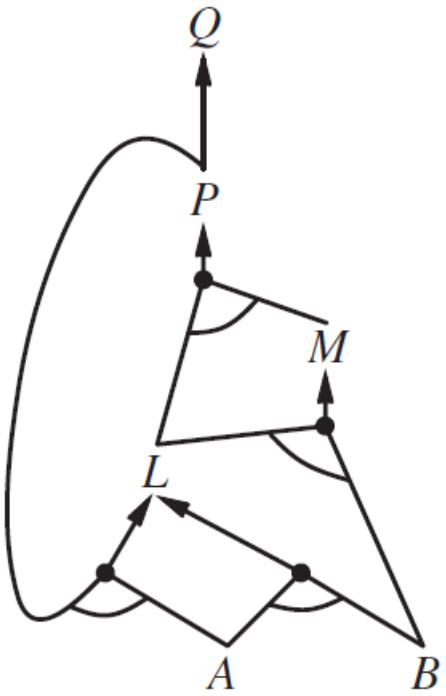
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

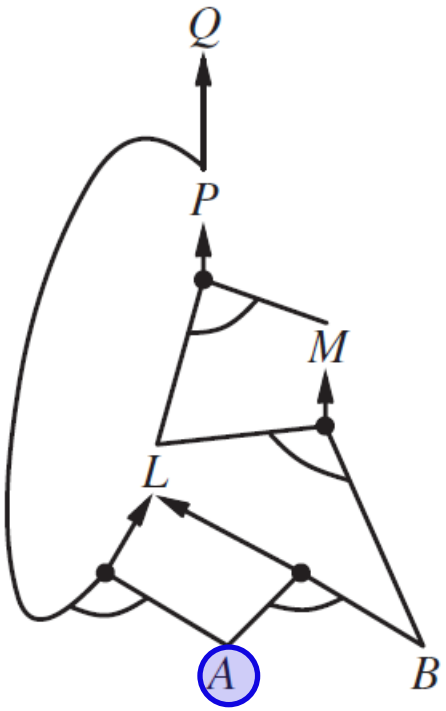
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

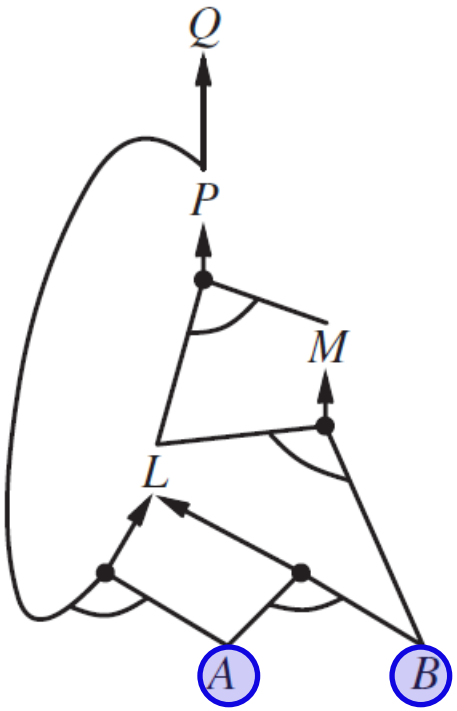


Execution

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$

A

B



Execution

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

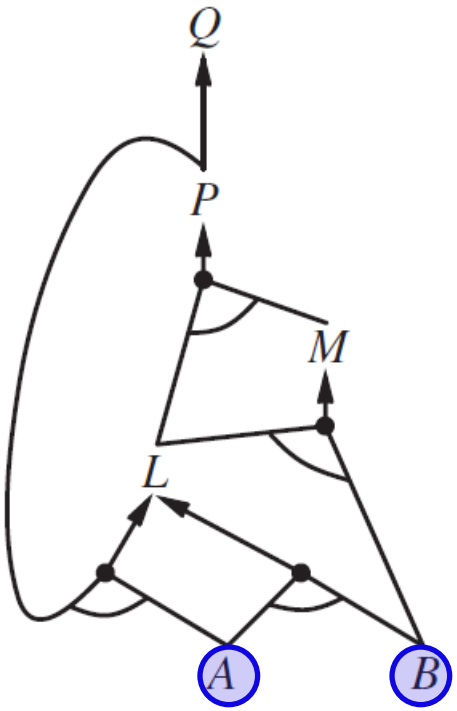
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$\boxed{A \wedge B \Rightarrow L}$$

$$A$$

$$B$$



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

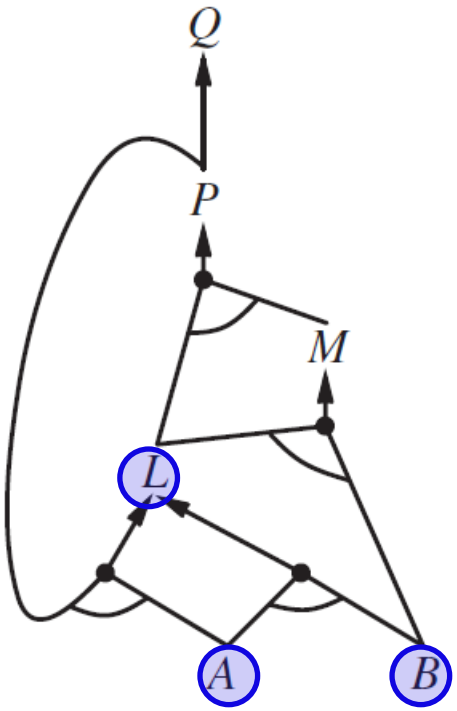
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

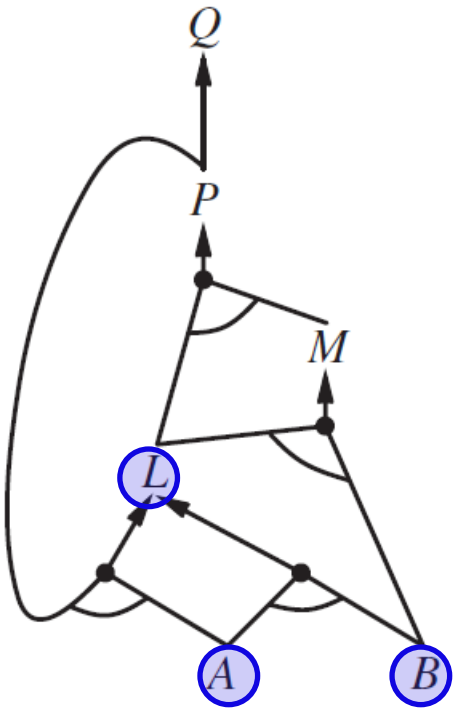
$$\boxed{B \wedge L \Rightarrow M}$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Execution

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

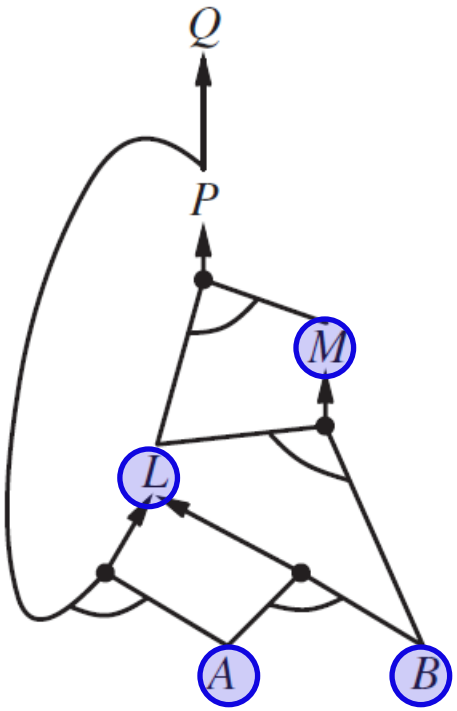
$$\boxed{B \wedge L \Rightarrow M}$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

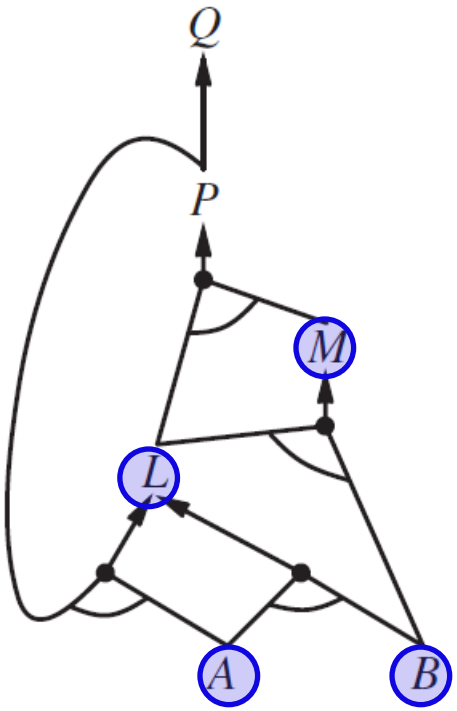
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

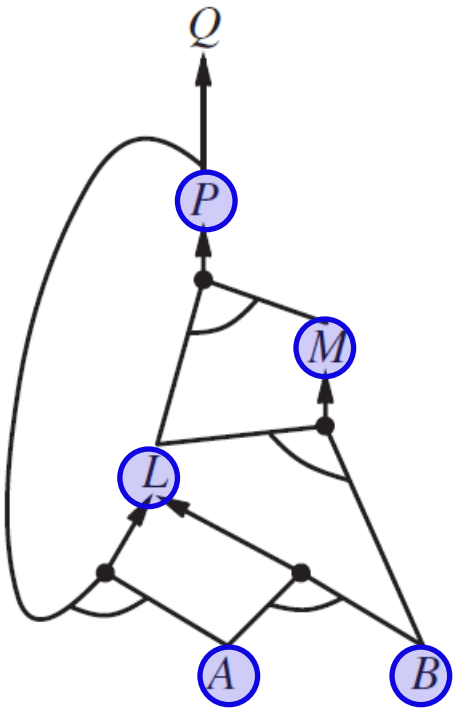
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$$\boxed{P \Rightarrow Q}$$

$$L \wedge M \Rightarrow P$$

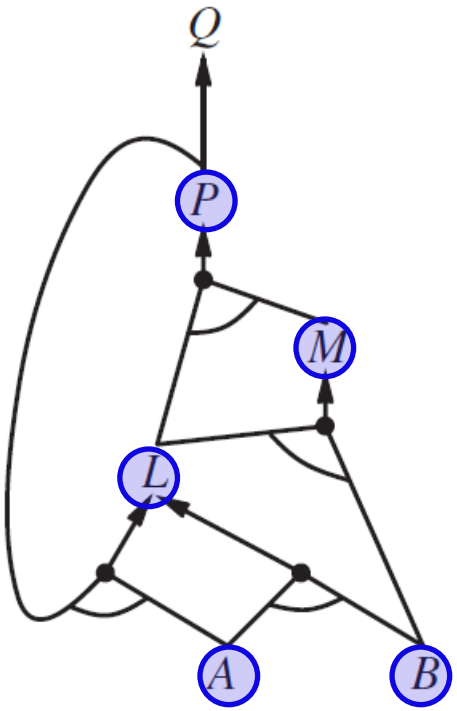
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Execution

$$\boxed{P \Rightarrow Q}$$

$$L \wedge M \Rightarrow P$$

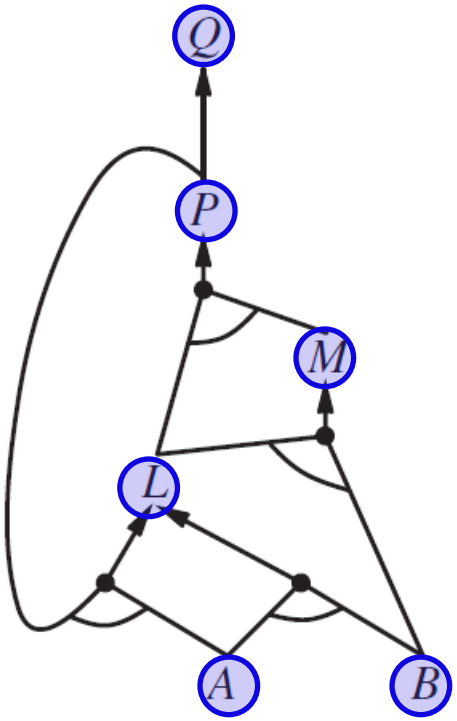
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Execution

$\boxed{P \Rightarrow Q}$

$L \wedge M \Rightarrow P$

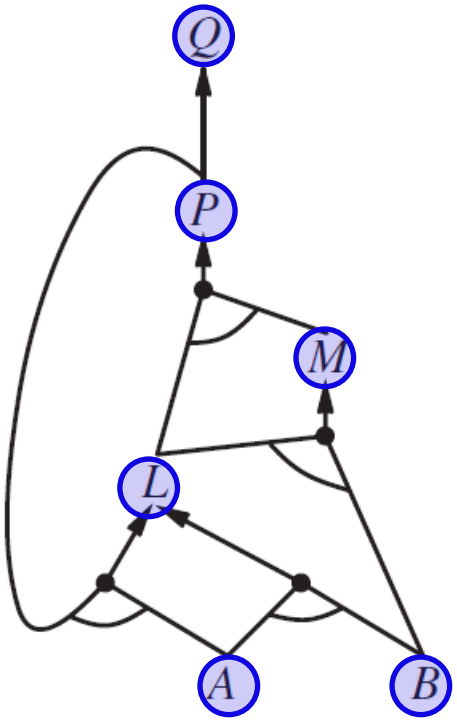
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Soundness of forward chaining: every inference is an application of Modus Ponens.

Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

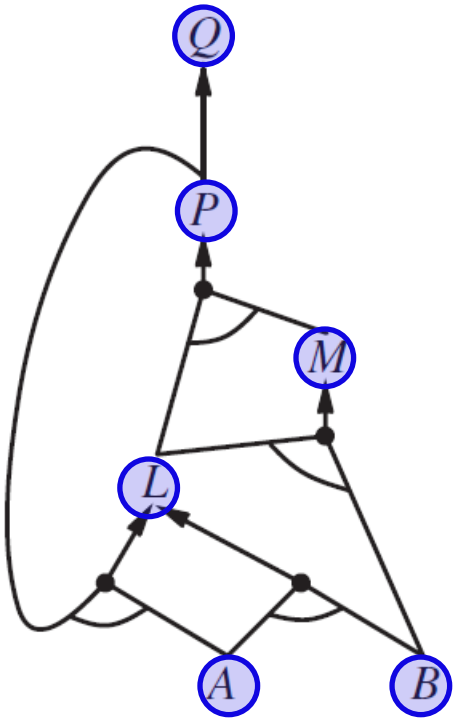
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Soundness of forward chaining: every inference is an application of Modus Ponens.

Completeness: every entailed atomic sentences will be derived.

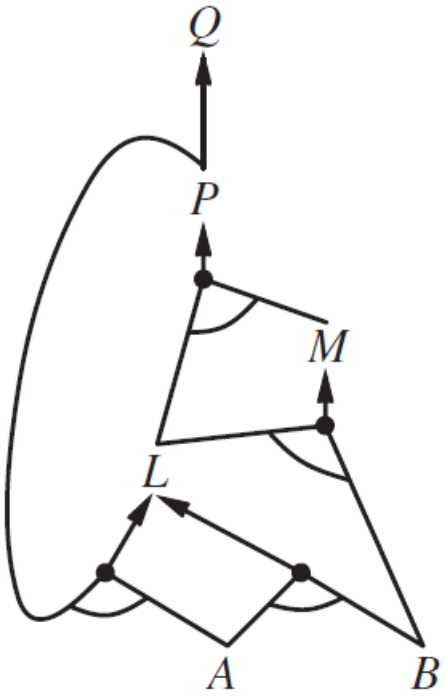
Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

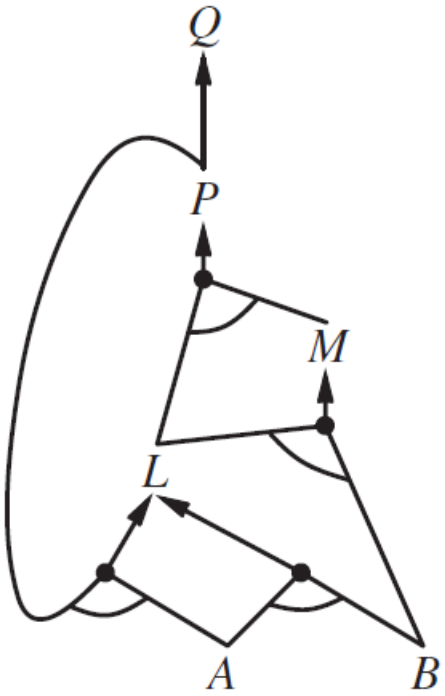


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

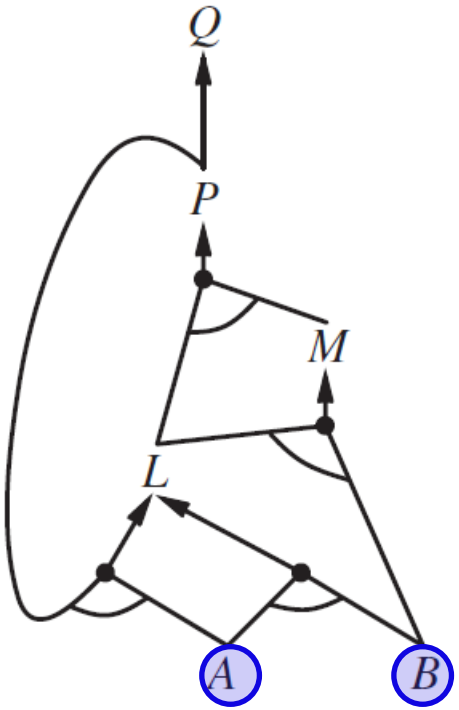


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

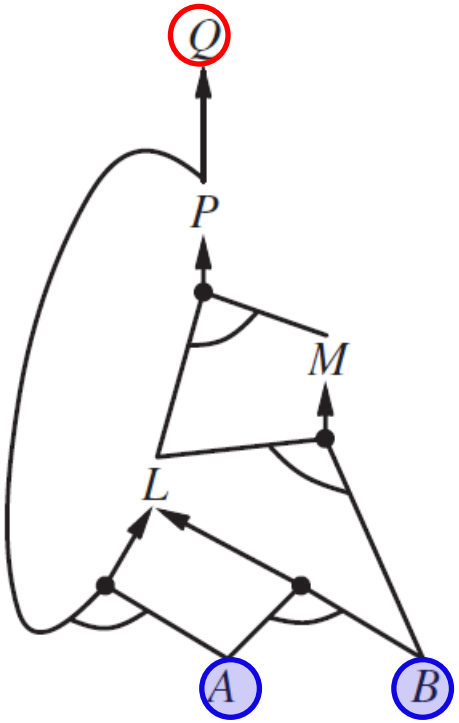


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$$\boxed{P \Rightarrow Q}$$

$$L \wedge M \Rightarrow P$$

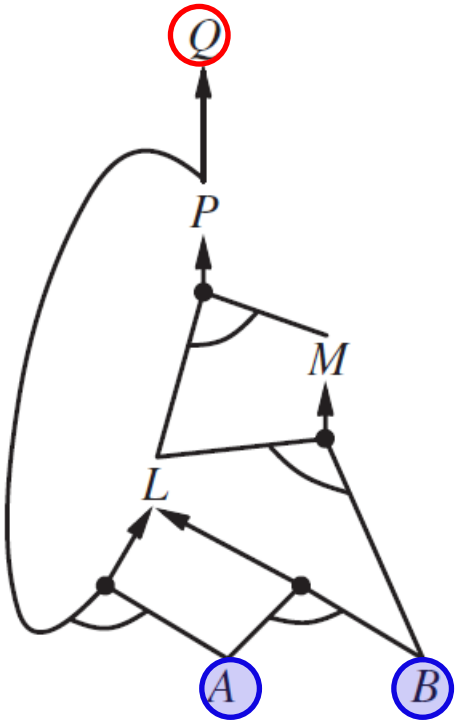
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$$\boxed{P \Rightarrow Q}$$

$$L \wedge M \Rightarrow P$$

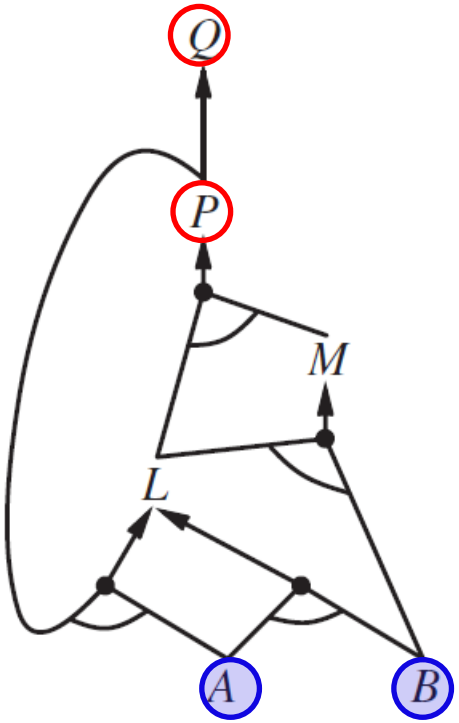
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

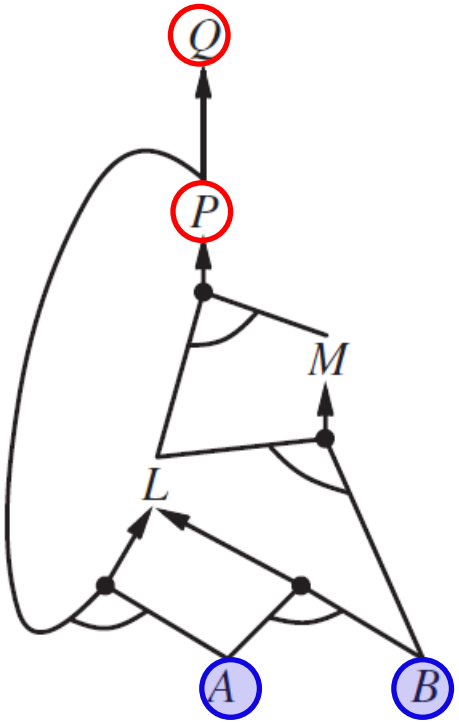
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

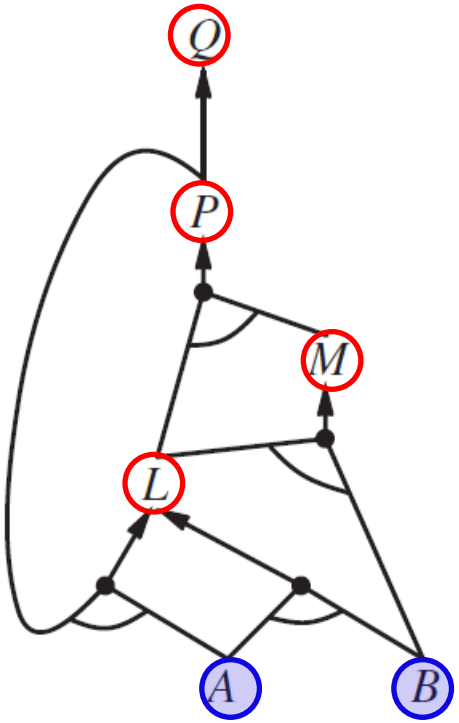
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

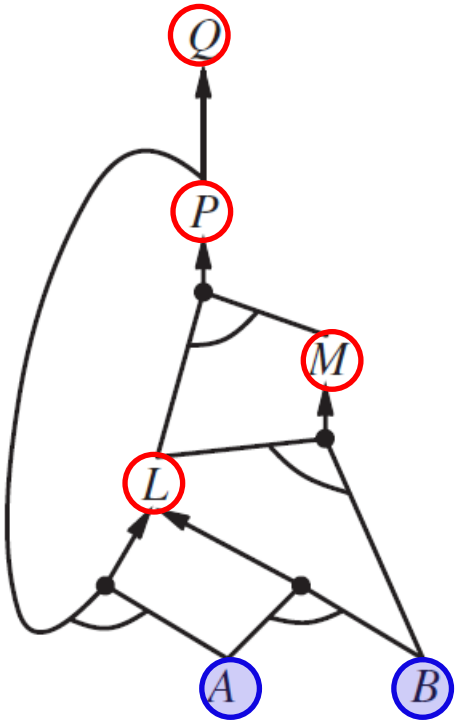
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

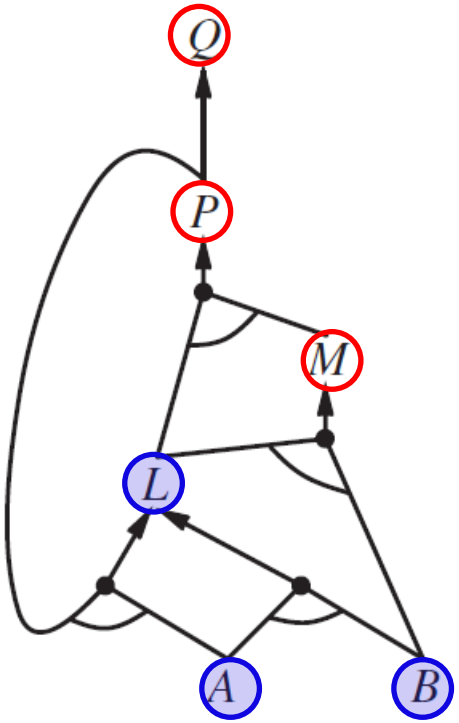
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

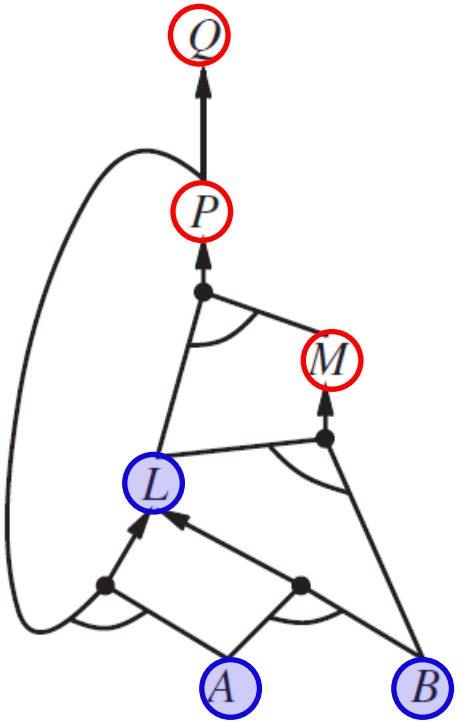
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

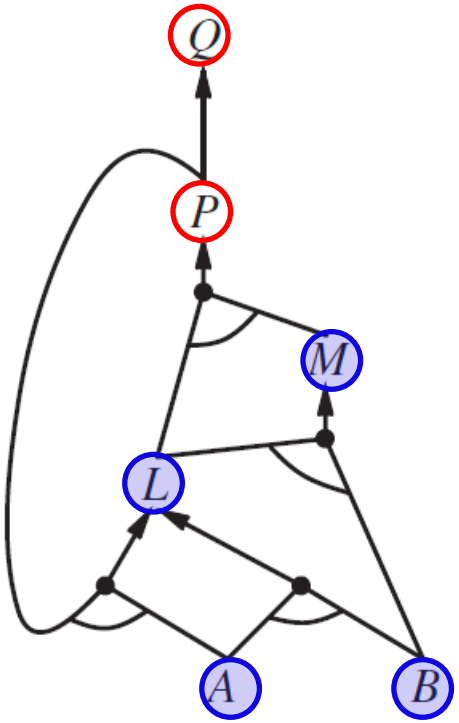
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

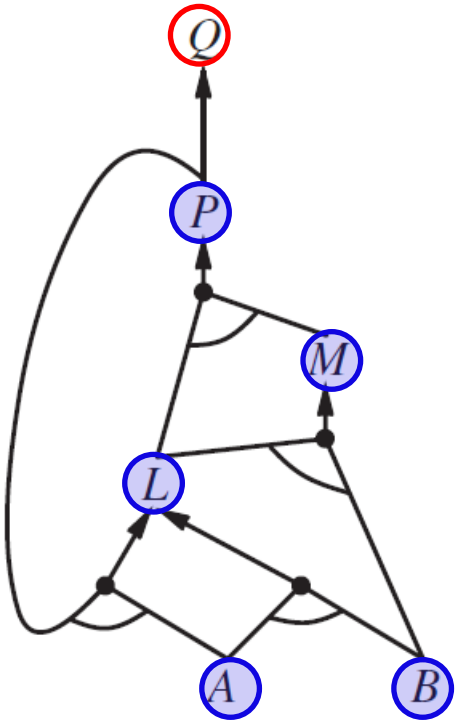
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

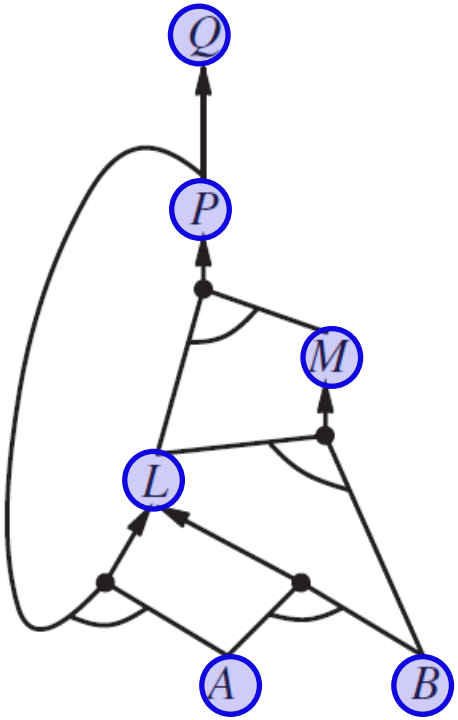


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. $KB \models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

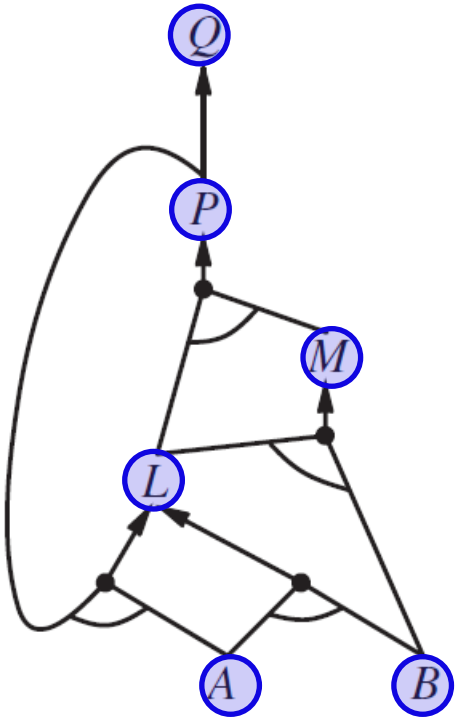
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



AND-OR graph search!

Forward vs. Backward Chaing

- Applicable range
  - Prove the entailment of a single proposition symbol
  - KB consists of definite clauses only.

Either P or $P_1 \wedge P_2 \wedge \cdots P_k \Rightarrow Q$

- Forward chaining is *data-driven*, automatic, unconscious processing.
- It may perform a lot of work that is irrelevant to the goal.
- Backward chaining is *goal-driven*, and appropriate for problem solving.
- It may run in time sublinear in the size of KB, since it touches only relevant facts.

IV. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

IV. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

One sentence in propositional logic (PL)

IV. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

One sentence in propositional logic (PL)

Satisfiability problem Is a sentence s in PL satisfiable?

IV. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

One sentence in propositional logic (PL)

Satisfiability problem Is a sentence s in PL satisfiable?

- Cast the problem as one of constraint satisfaction.

Many combinatorial problems in computer science can be reduced to checking the satisfiability of a propositional sentence.

IV. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

One sentence in propositional logic (PL)

Satisfiability problem Is a sentence s in PL satisfiable?

- Cast the problem as one of constraint satisfaction.

  Many combinatorial problems in computer science can be reduced to checking the satisfiability of a propositional sentence.

- Complete backtracking search (DPLL algorithm)

- Incomplete local search (WALKSAT algorithm)

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
        DPLL($clauses, rest, model \cup \{P{=}false\}$))

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
        DPLL($clauses, rest, model \cup \{P{=}false\}$))

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
      DPLL($clauses, rest, model \cup \{P{=}false\}$))

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., A and B are pure in $A \vee \neg B$, $\neg B \vee \neg C$, $C \vee A$ while C is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \vee \neg C$, enabling C to become a pure symbol.

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

Truth value to assign to P

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
        DPLL($clauses, rest, model \cup \{P{=}false\}$))

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., A and B are pure in $A \vee \neg B$, $\neg B \vee \neg C$, $C \vee A$ while C is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \vee \neg C$, enabling C to become a pure symbol.

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

Truth value to assign to P

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

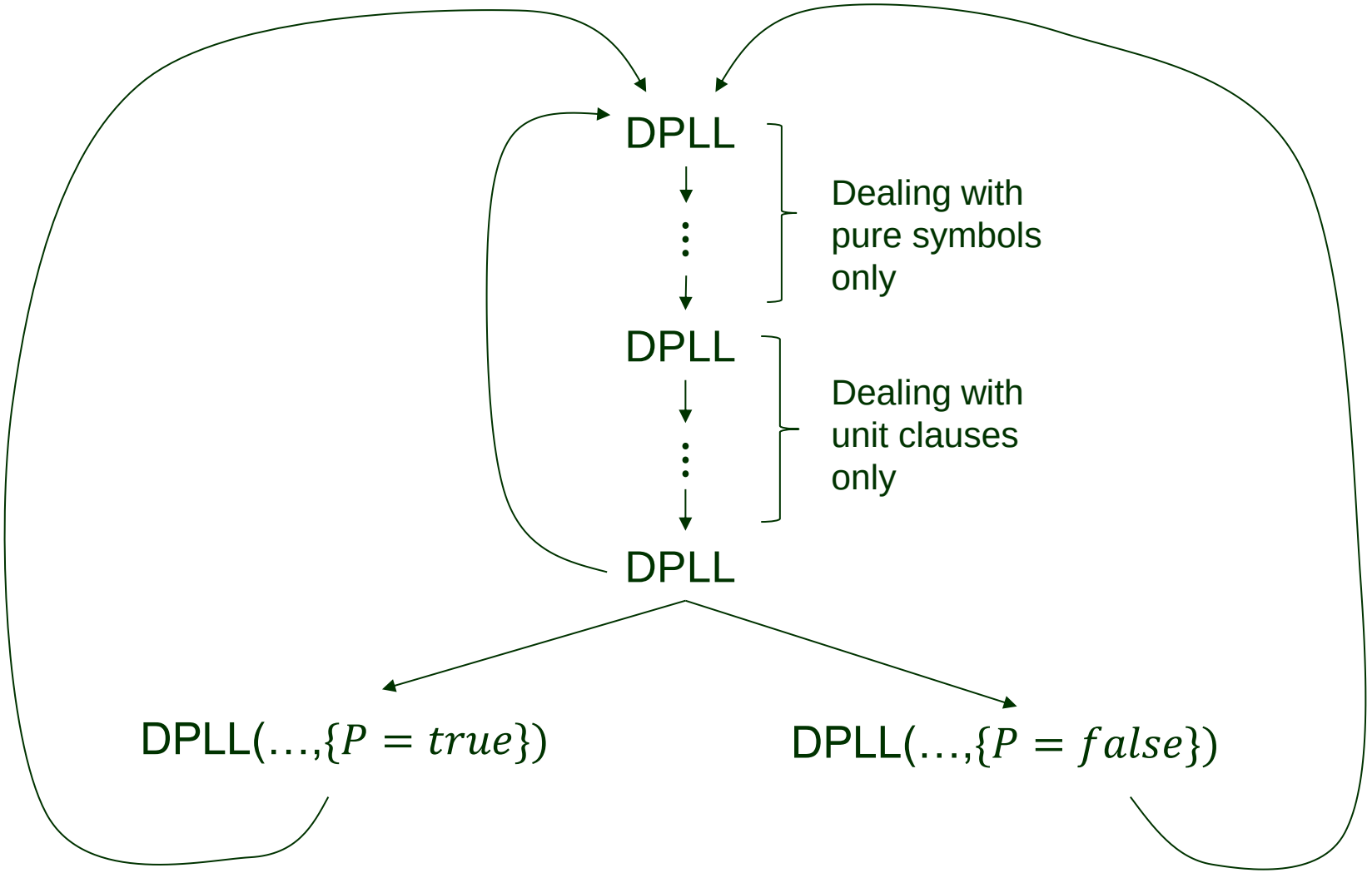
  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
    DPLL($clauses, rest, model \cup \{P{=}false\}$))

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., A and B are pure in $A \vee \neg B$, $\neg B \vee \neg C$, $C \vee A$ while C is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \vee \neg C$, enabling C to become a pure symbol.

Unit clause propagation on a clause in which all literals but one are assigned *false*. E.g., $\neg B \vee \neg C$ simplifies to the unit clause $\neg C$ if B = *true*.

Recursion Tree



Example (Exercise 7.25)

S_1: $A \Leftrightarrow (B \vee E)$

Convertion into clauses

S_2: $E \Rightarrow D$

S_3: $C \wedge F \Rightarrow \neg B$

S_4: $E \Rightarrow B$

S_5: $B \Rightarrow F$

S_6: $B \Rightarrow C$

Example (Exercise 7.25)

S_1: $A \Leftrightarrow (B \vee E)$

$\Downarrow$

$\big(A \Rightarrow (B \vee E)\big) \wedge \big((B \vee E) \Rightarrow A\big)$

$\Downarrow$

$(\neg A \vee B \vee E) \wedge (\neg(B \vee E) \vee A)$

$\Downarrow$

$(\neg A \vee B \vee E) \wedge ((\neg B \wedge \neg E) \vee A)$

Convertion into clauses

S_2: $E \Rightarrow D$

S_3: $C \wedge F \Rightarrow \neg B$

S_4: $E \Rightarrow B$

S_5: $B \Rightarrow F$

S_6: $B \Rightarrow C$

Example (Exercise 7.25)

S_1: $A \Leftrightarrow (B \vee E)$

$\Downarrow$

$\big(A \Rightarrow (B \vee E)\big) \wedge \big((B \vee E) \Rightarrow A\big)$

$\Downarrow$

$(\neg A \vee B \vee E) \wedge (\neg(B \vee E) \vee A)$

$\Downarrow$

$(\neg A \vee B \vee E) \wedge ((\neg B \wedge \neg E) \vee A)$

C_1: $\neg A \vee B \vee E$

C_2: $\neg B \vee A$

C_3: $\neg E \vee A$

Convertion into clauses

S_2: $E \Rightarrow D$

S_3: $C \wedge F \Rightarrow \neg B$

S_4: $E \Rightarrow B$

S_5: $B \Rightarrow F$

S_6: $B \Rightarrow C$

Example (Exercise 7.25)

S_1: $A \Leftrightarrow (B \vee E)$

$\Downarrow$

$\big(A \Rightarrow (B \vee E)\big) \wedge \big((B \vee E) \Rightarrow A\big)$

$\Downarrow$

$(\neg A \vee B \vee E) \wedge (\neg(B \vee E) \vee A)$

$\Downarrow$

$(\neg A \vee B \vee E) \wedge ((\neg B \wedge \neg E) \vee A)$

C_1: $\neg A \vee B \vee E$

C_2: $\neg B \vee A$

C_3: $\neg E \vee A$

Convertion into clauses

S_2: $E \Rightarrow D$ $\Rightarrow$ C_4: $\neg E \vee D$

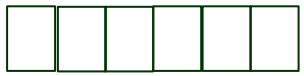
S_3: $C \wedge F \Rightarrow \neg B$ $\Rightarrow$ C_5: $\neg C \vee \neg F \vee \neg B$

S_4: $E \Rightarrow B$ $\Rightarrow$ C_6: $\neg E \vee B$

S_5: $B \Rightarrow F$ $\Rightarrow$ C_7: $\neg B \vee F$

S_6: $B \Rightarrow C$ $\Rightarrow$ C_8: $\neg B \vee C$

A B C D E F

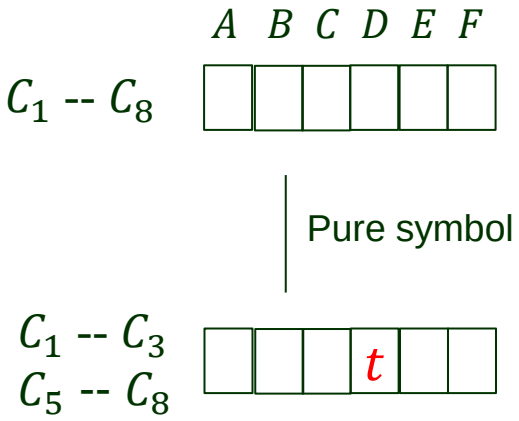
C_1 -- C_8 

C_1: $\neg A \vee B \vee E$ C_2: $\neg B \vee A$

C_3: $\neg E \vee A$ C_4: $\neg E \vee D$

C_5: $\neg C \vee \neg F \vee \neg B$ C_6: $\neg E \vee B$

C_7: $\neg B \vee F$ C_8: $\neg B \vee C$

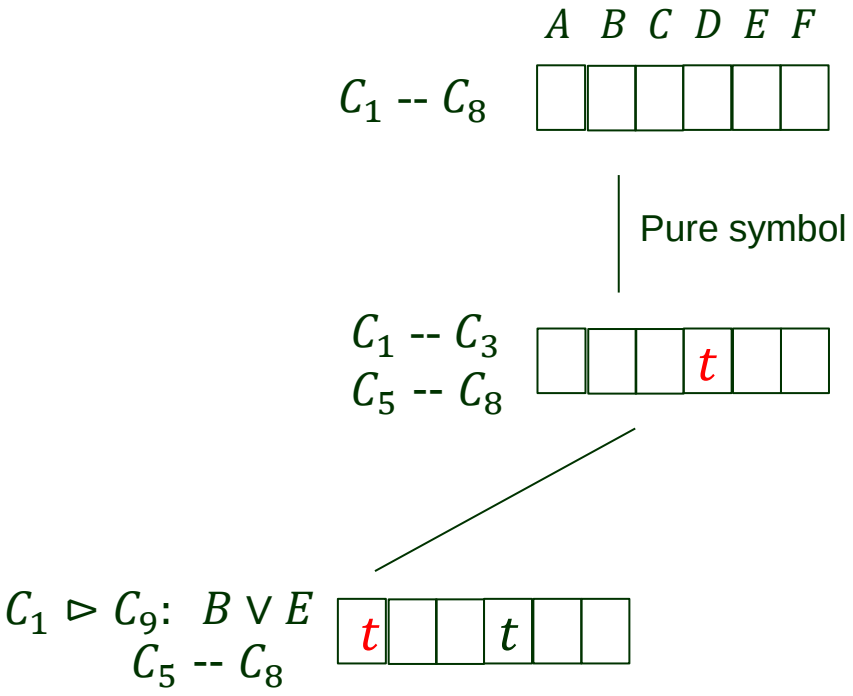


C_1: $\neg A \vee B \vee E$ C_2: $\neg B \vee A$

C_3: $\neg E \vee A$ C_4: $\neg E \vee D$

C_5: $\neg C \vee \neg F \vee \neg B$ C_6: $\neg E \vee B$

C_7: $\neg B \vee F$ C_8: $\neg B \vee C$

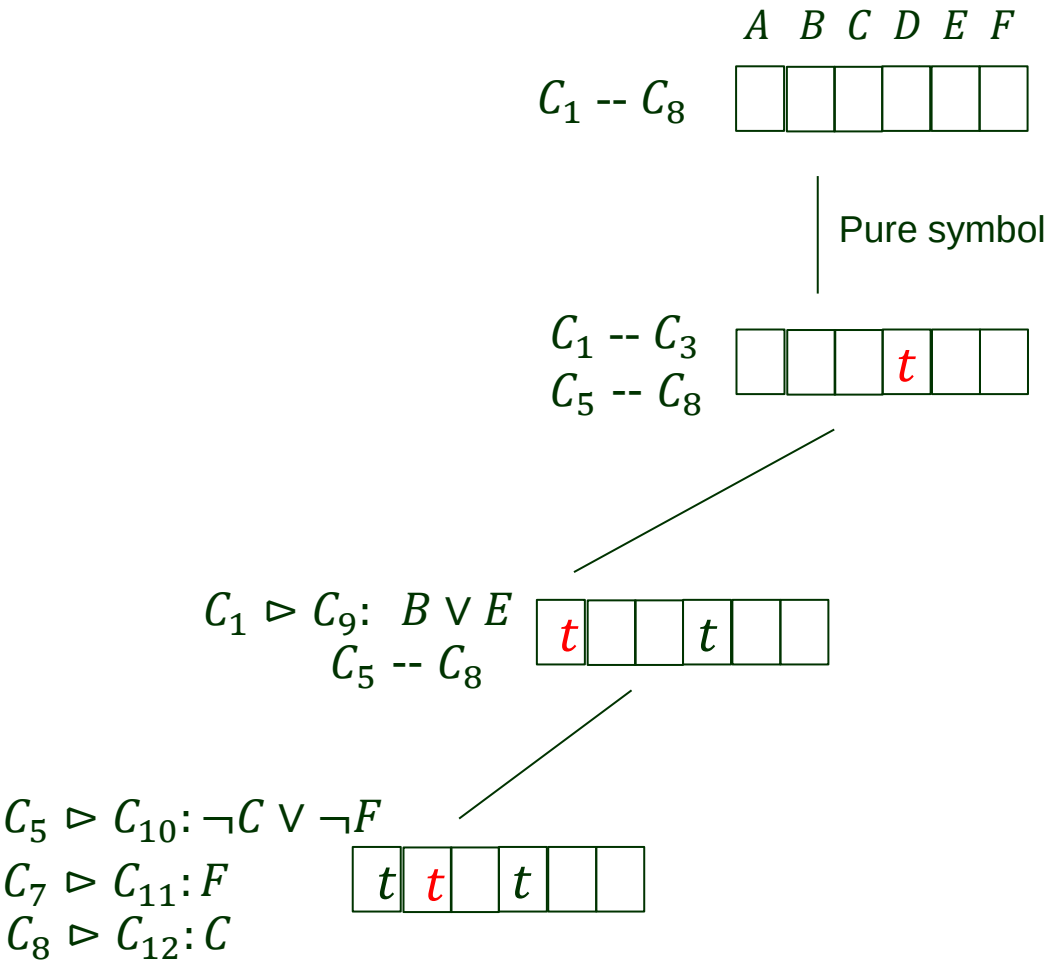


C_1: $\neg A \vee B \vee E$ C_2: $\neg B \vee A$

C_3: $\neg E \vee A$ C_4: $\neg E \vee D$

C_5: $\neg C \vee \neg F \vee \neg B$ C_6: $\neg E \vee B$

C_7: $\neg B \vee F$ C_8: $\neg B \vee C$

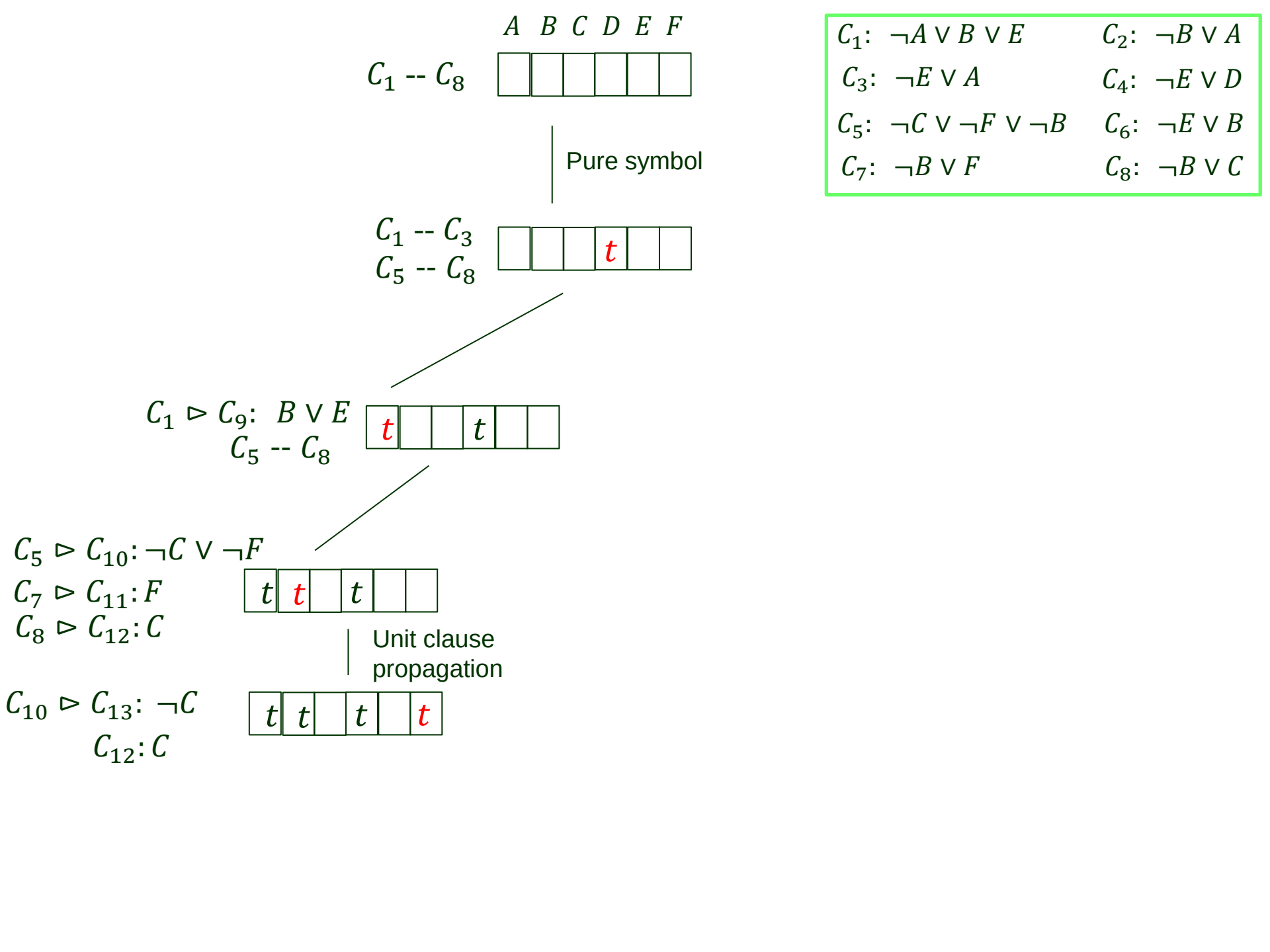


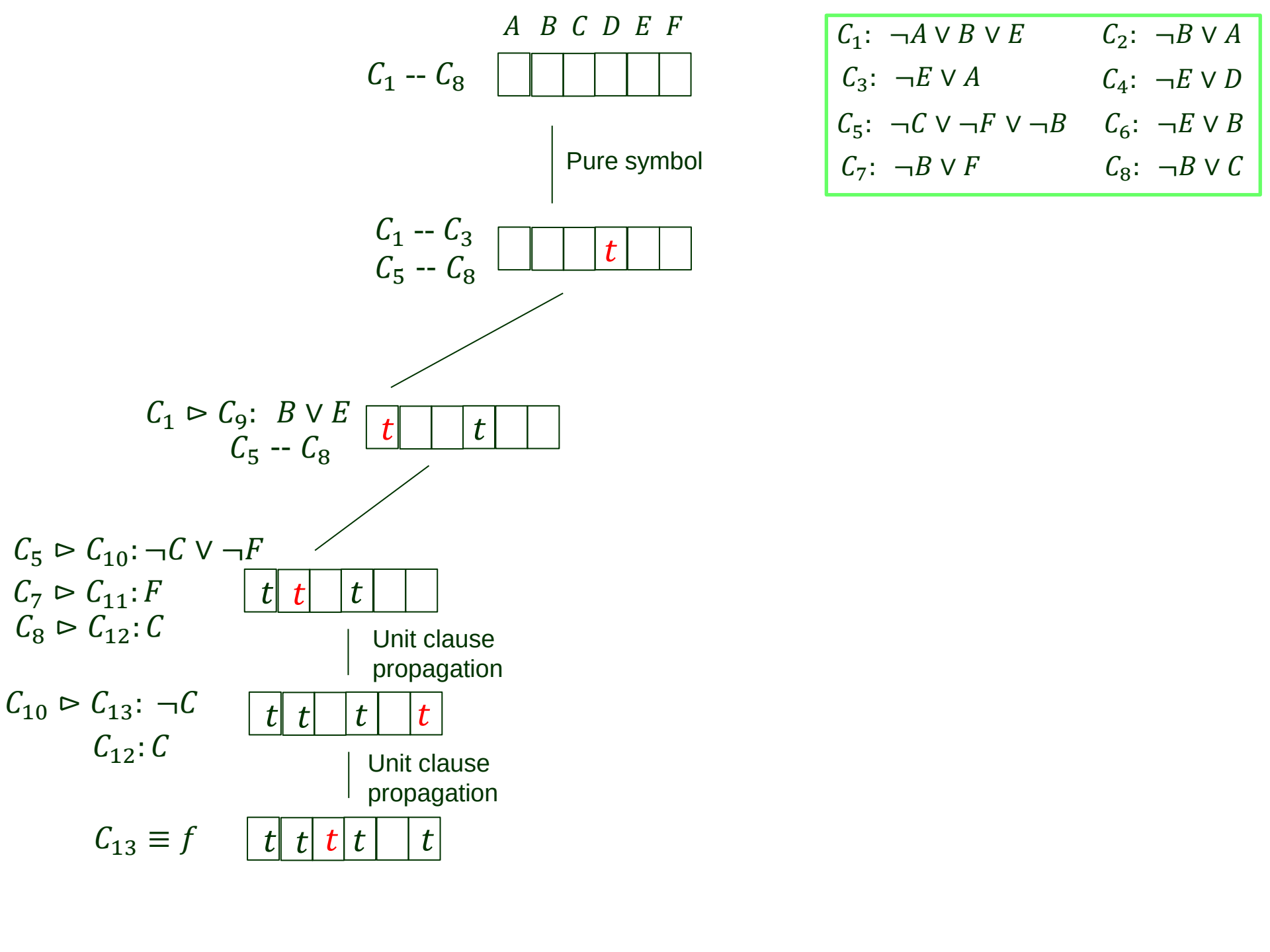
C_1: $\neg A \vee B \vee E$ C_2: $\neg B \vee A$

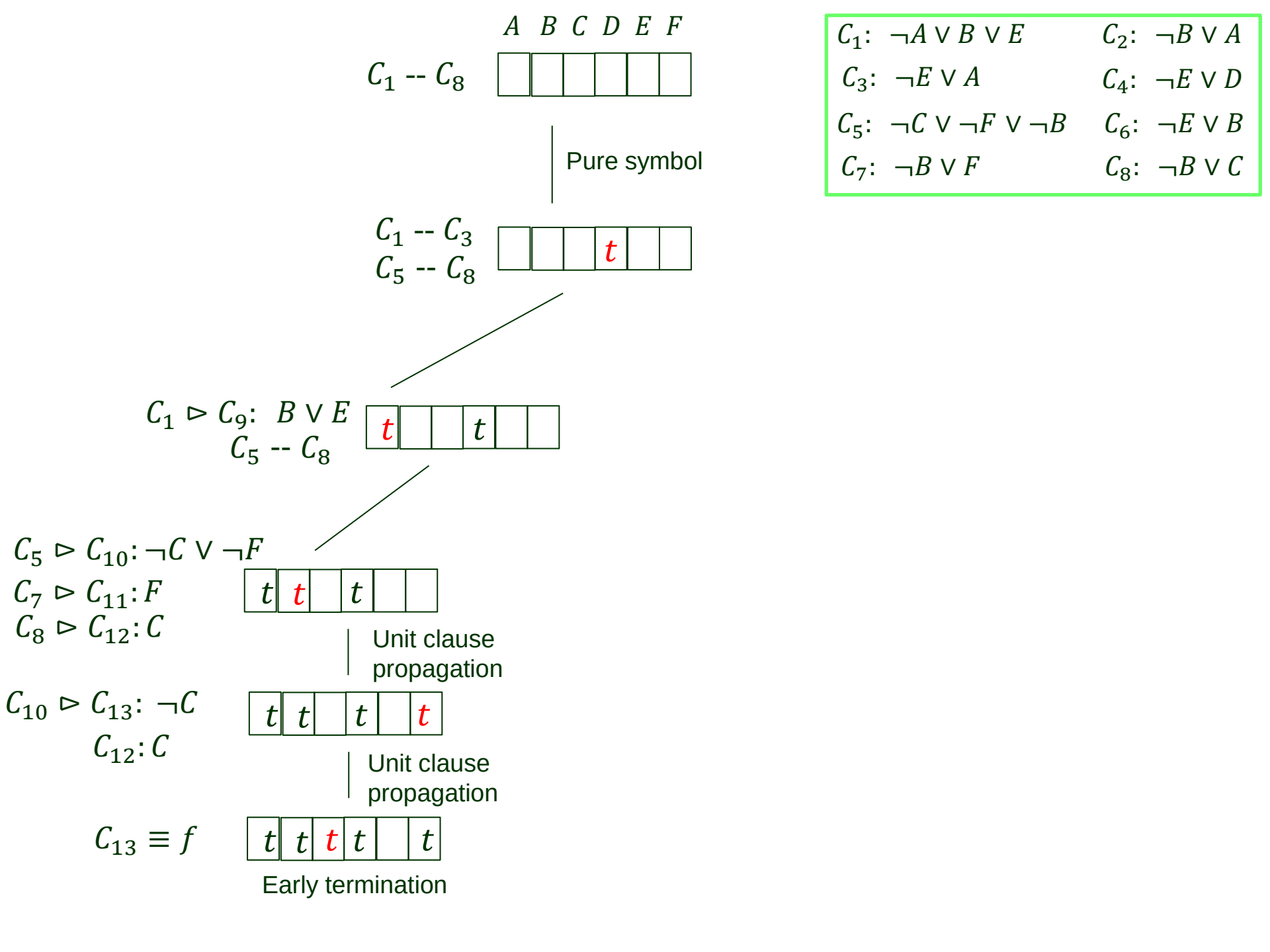
C_3: $\neg E \vee A$ C_4: $\neg E \vee D$

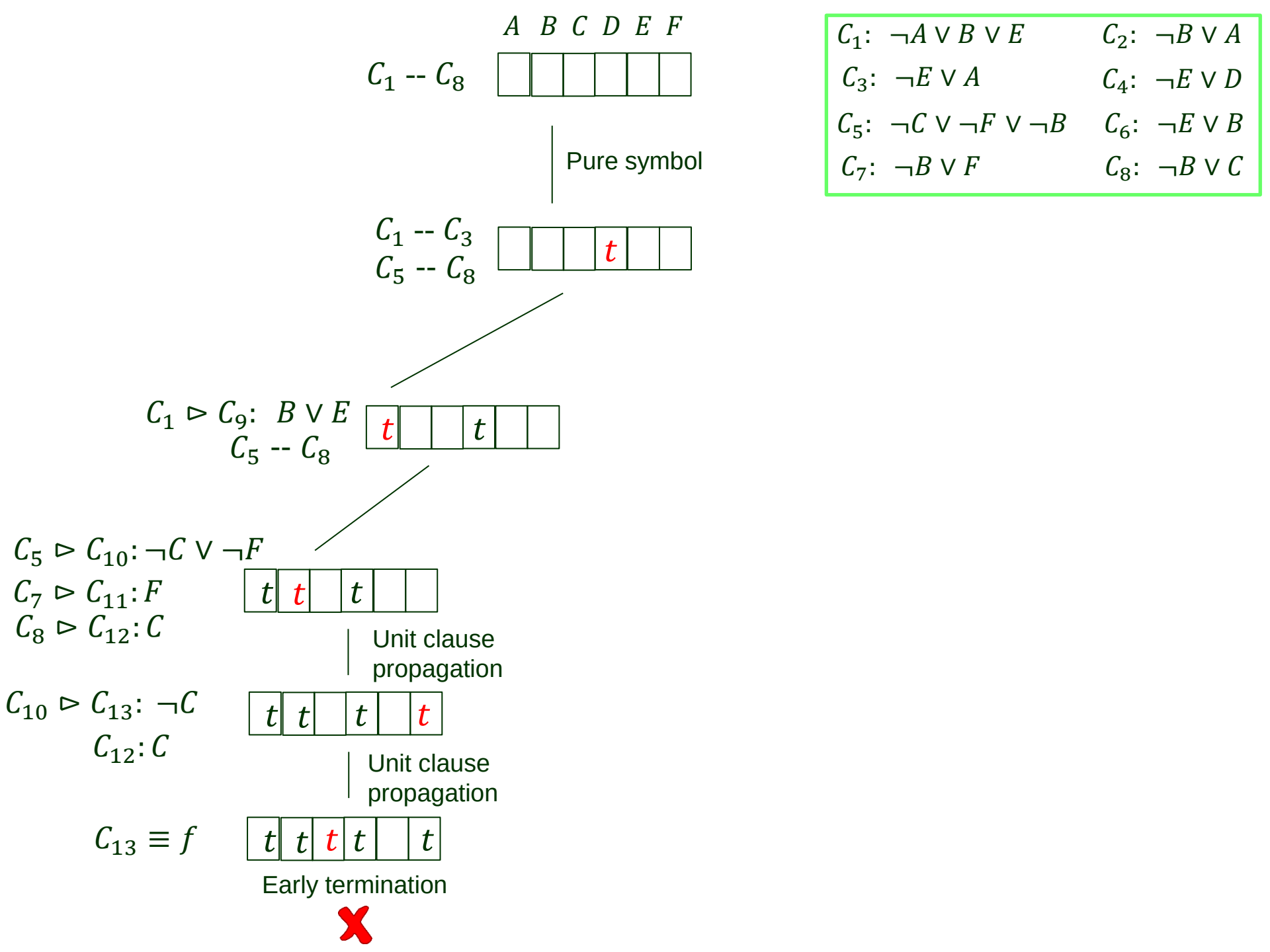
C_5: $\neg C \vee \neg F \vee \neg B$ C_6: $\neg E \vee B$

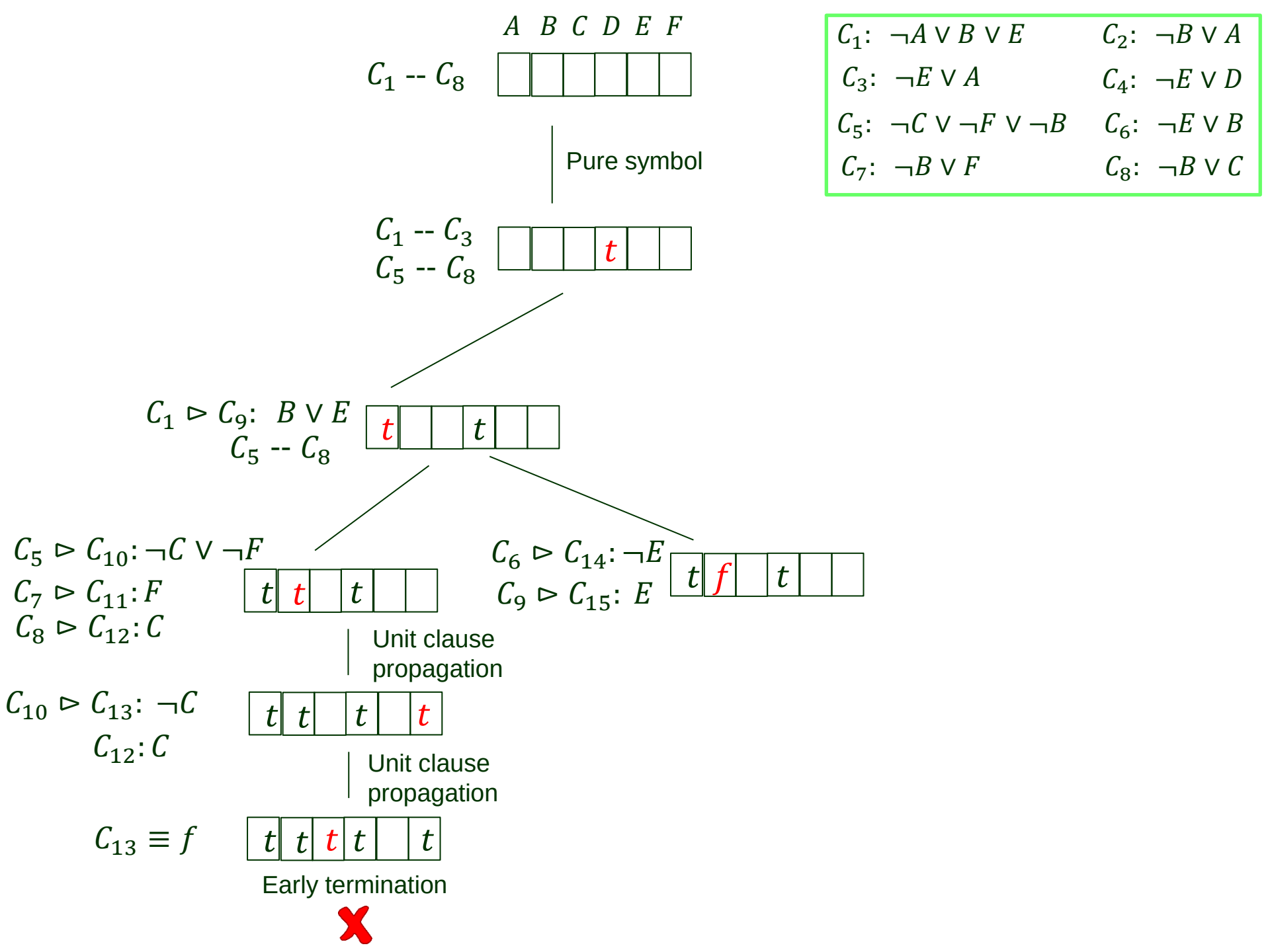
C_7: $\neg B \vee F$ C_8: $\neg B \vee C$

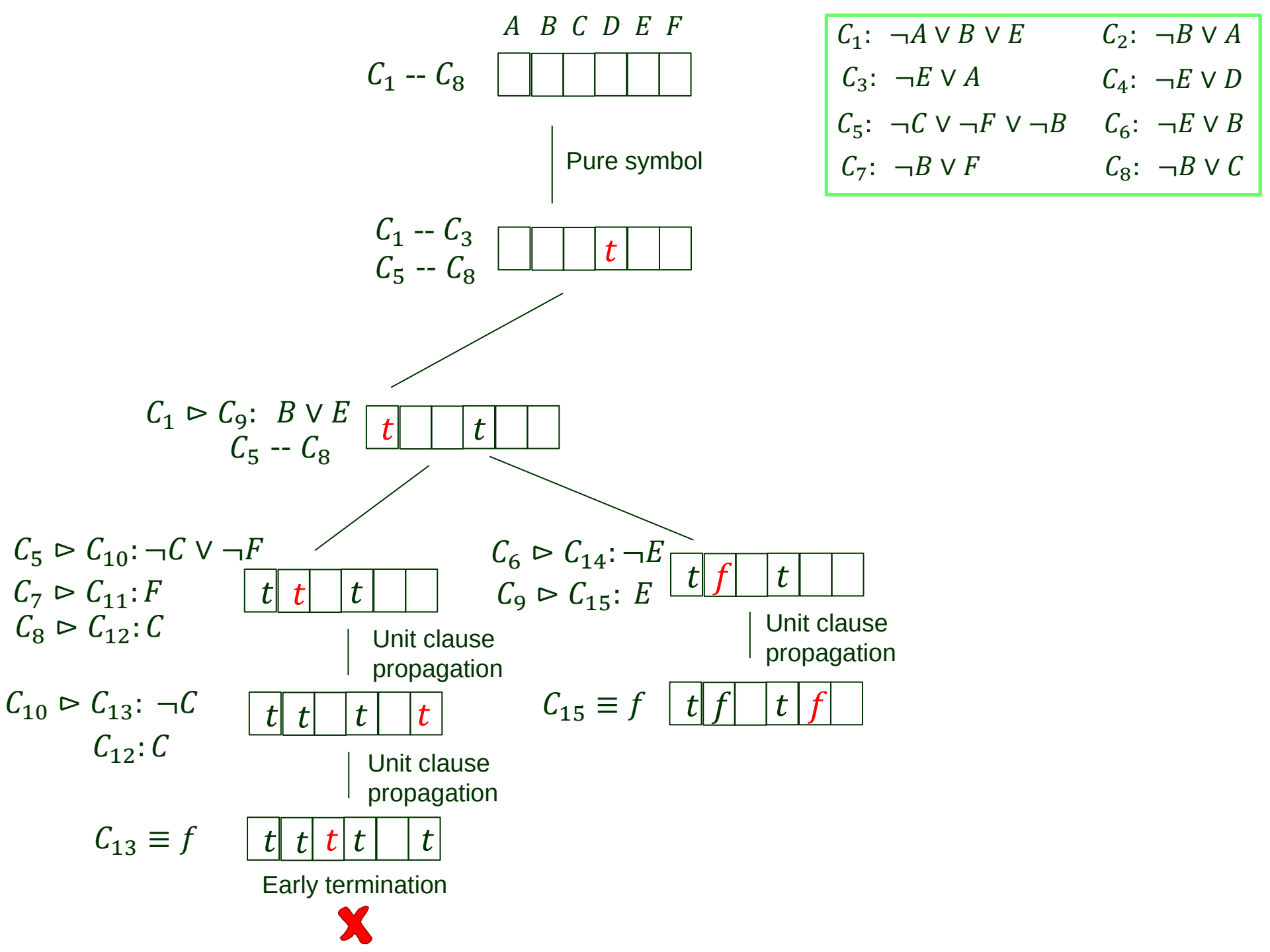

A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t
C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$

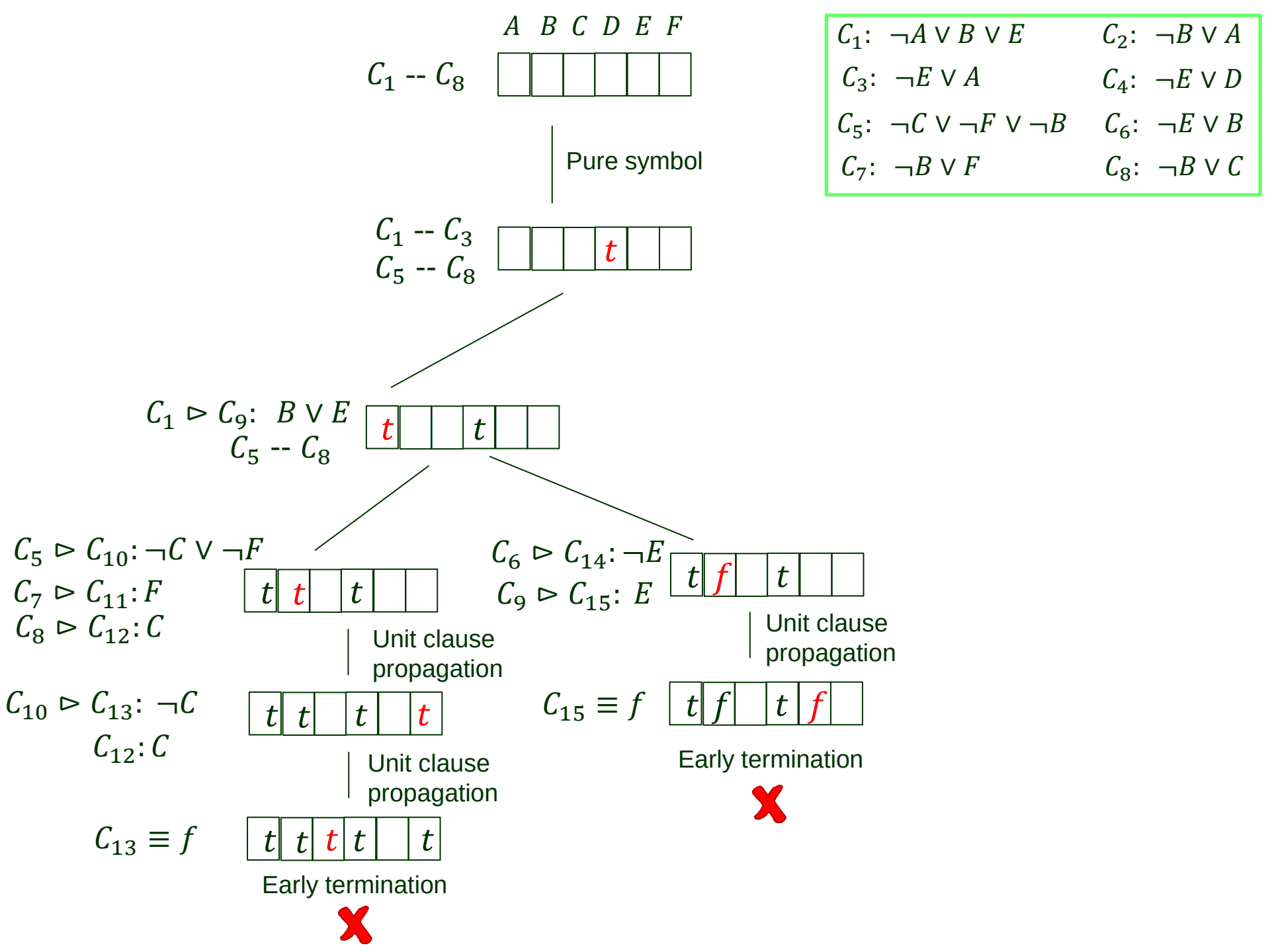

C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t

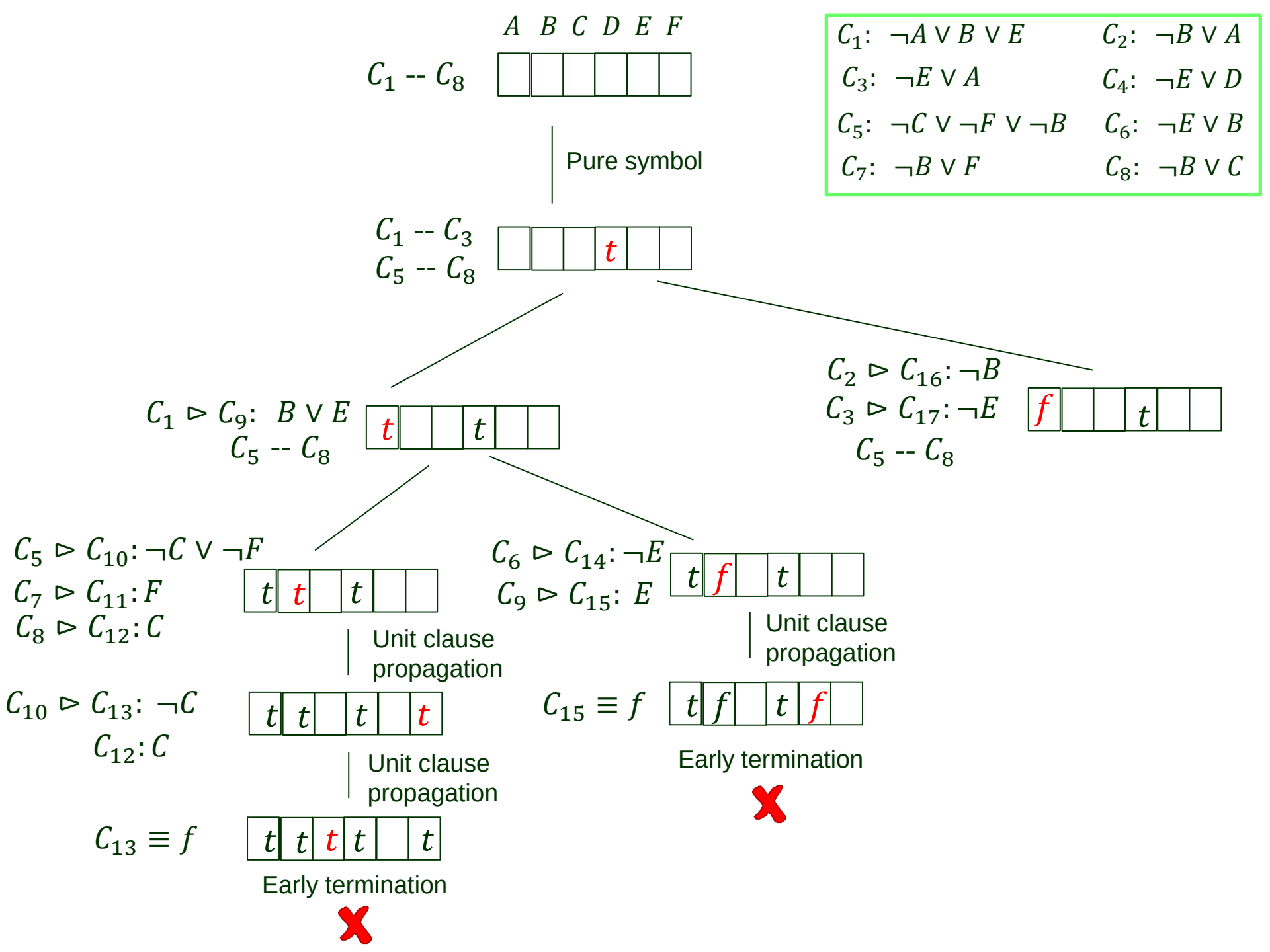

C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t
Early termination

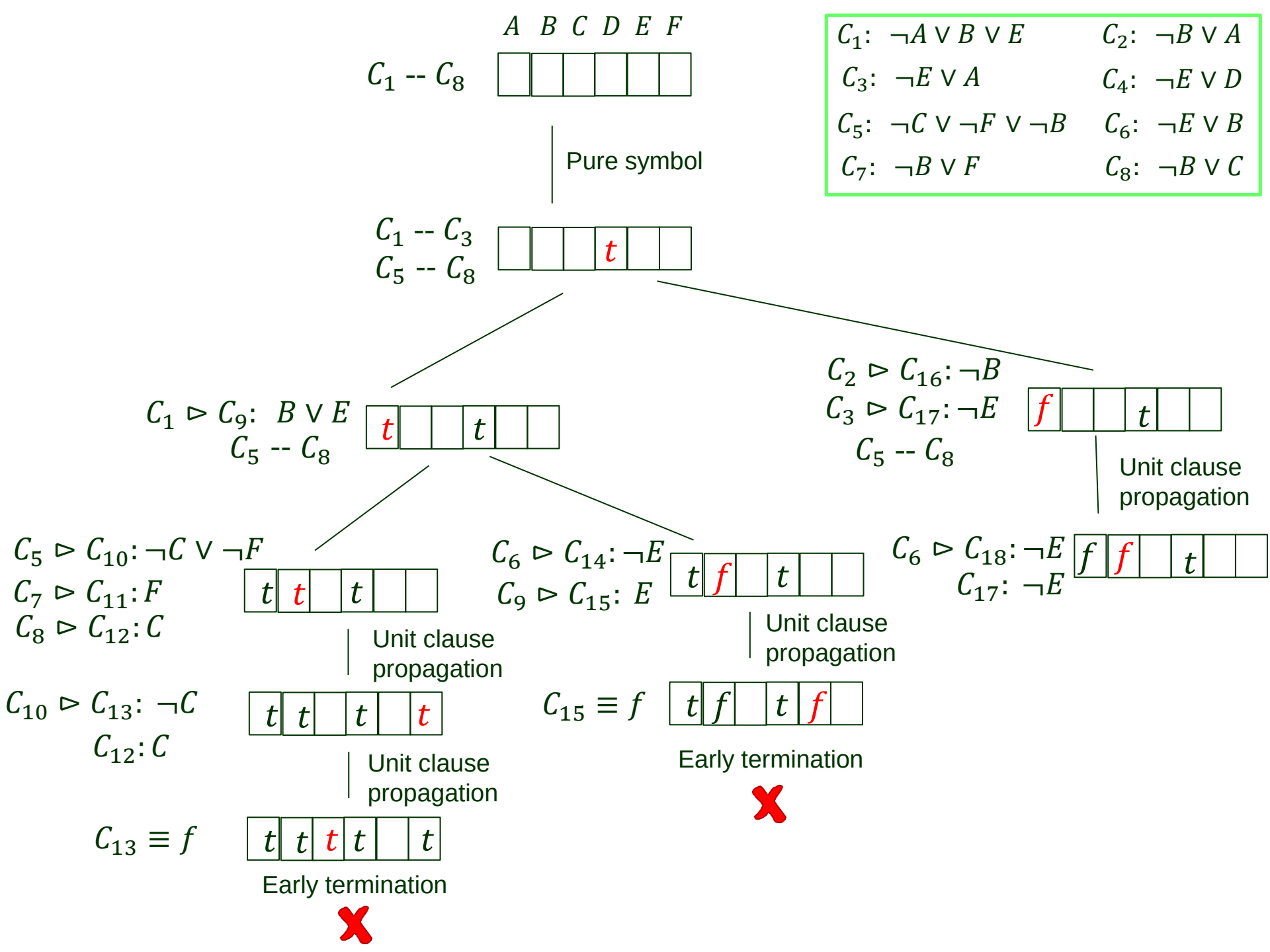

C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t
Early termination

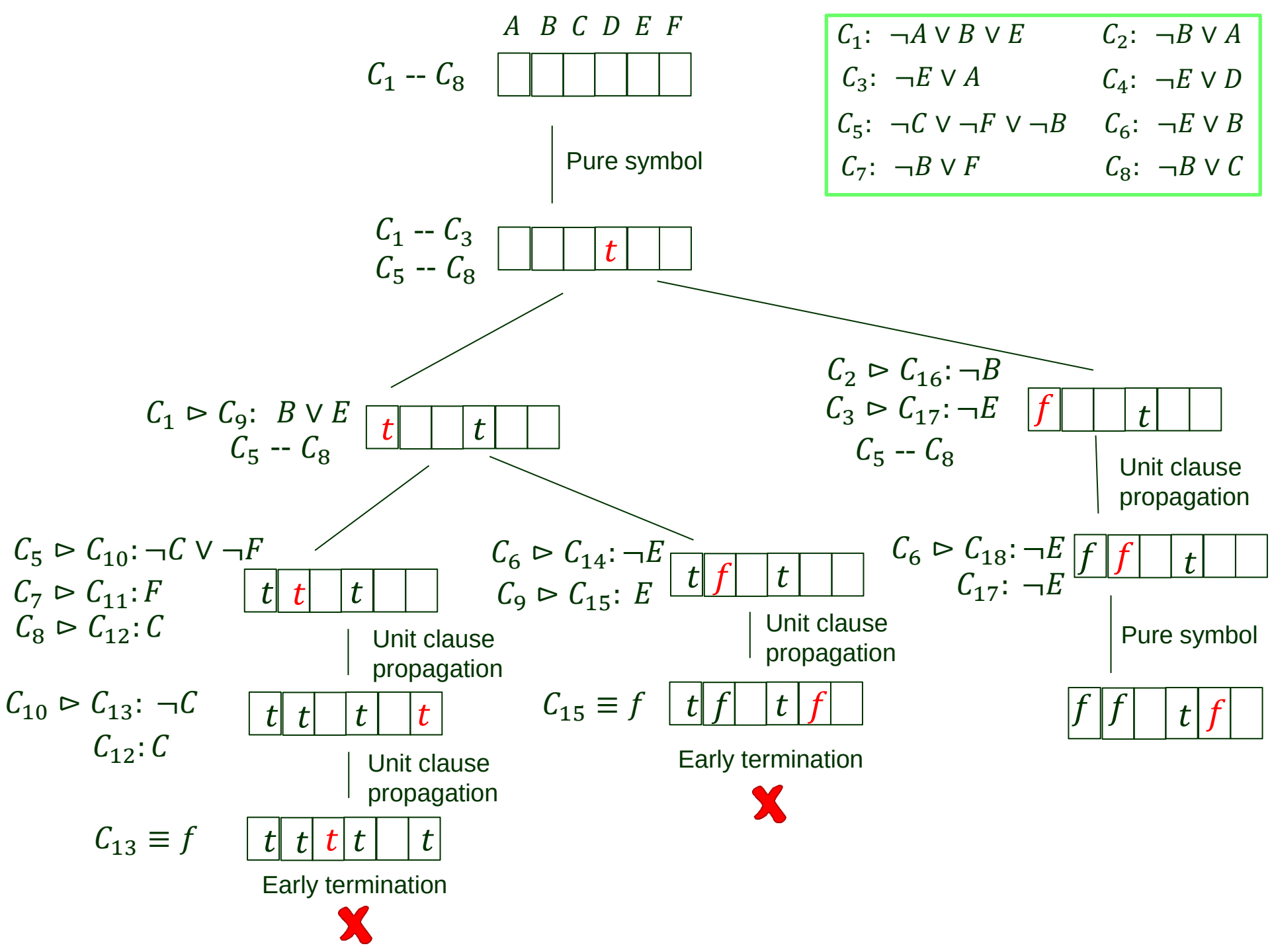

C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
t f t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t
Early termination

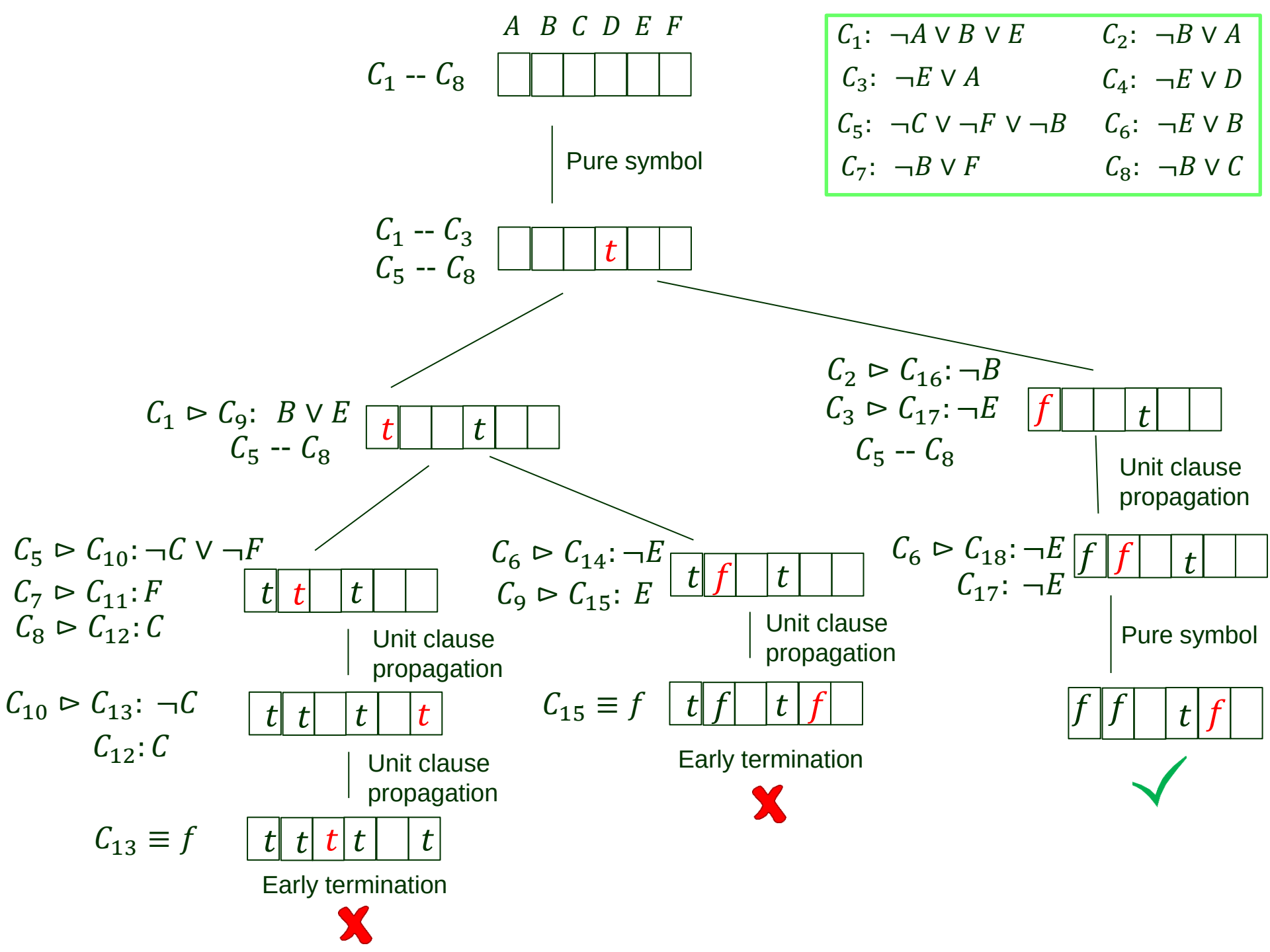

C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t
Early termination
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
t f t
Unit clause propagation
$C_{15} \equiv f$
t f t f


C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t
Early termination
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
t f t
Unit clause propagation
$C_{15} \equiv f$
t f t f
Early termination


C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
$C_2 \triangleright C_{16}$: $\neg B$
$C_3 \triangleright C_{17}$: $\neg E$
C_5 -- C_8
f
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
Unit clause propagation
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
$C_{15} \equiv f$
Unit clause propagation
Early termination
$C_{13} \equiv f$
Early termination


C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
$C_2 \triangleright C_{16}$: $\neg B$
$C_3 \triangleright C_{17}$: $\neg E$
C_5 -- C_8
f
Unit clause propagation
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
$C_6 \triangleright C_{18}$: $\neg E$
C_{17}: $\neg E$
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
$C_{15} \equiv f$
Early termination
$C_{13} \equiv f$
Early termination


C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
$C_2 \triangleright C_{16}$: $\neg B$
$C_3 \triangleright C_{17}$: $\neg E$
C_5 -- C_8
f
Unit clause propagation
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
$C_6 \triangleright C_{18}$: $\neg E$
C_{17}: $\neg E$
Unit clause propagation
Unit clause propagation
Pure symbol
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
$C_{15} \equiv f$
Unit clause propagation
Early termination
$C_{13} \equiv f$
Early termination


C_1: $\neg A \vee B \vee E$
C_2: $\neg B \vee A$
C_3: $\neg E \vee A$
C_4: $\neg E \vee D$
C_5: $\neg C \vee \neg F \vee \neg B$
C_6: $\neg E \vee B$
C_7: $\neg B \vee F$
C_8: $\neg B \vee C$
A B C D E F
C_1 -- C_8
Pure symbol
C_1 -- C_3
C_5 -- C_8
t
$C_1 \triangleright C_9$: $B \vee E$
C_5 -- C_8
t t t
$C_2 \triangleright C_{16}$: $\neg B$
$C_3 \triangleright C_{17}$: $\neg E$
C_5 -- C_8
f t t
Unit clause propagation
$C_5 \triangleright C_{10}$: $\neg C \vee \neg F$
$C_7 \triangleright C_{11}$: F
$C_8 \triangleright C_{12}$: C
t t t t
Unit clause propagation
$C_{10} \triangleright C_{13}$: $\neg C$
C_{12}: C
t t t t t
Unit clause propagation
$C_{13} \equiv f$
t t t t t
Early termination
$C_6 \triangleright C_{14}$: $\neg E$
$C_9 \triangleright C_{15}$: E
t f t
Unit clause propagation
$C_{15} \equiv f$
t f t f
Early termination
$C_6 \triangleright C_{18}$: $\neg E$
C_{17}: $\neg E$
f f t
Pure symbol
f f t f

Local Search Algorithms

- Take steps in the space of complete assignments, flipping the truth value of one symbol at a time.
- Use an evaluation that counts the number of unsatisfied clauses.
- Escape local minima using various forms of randomness.
- Find a good balance between greediness and randomness.

The WALKSAT Algorithm

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of value flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for each $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    if RANDOM$(0, 1) \leq p$ **then**
      flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

The WALKSAT Algorithm

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of value flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for each $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    if RANDOM$(0, 1) \leq p$ **then**
      flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

The WALKSAT Algorithm

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of value flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for each $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    if RANDOM$(0, 1) \leq p$ **then**
      flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*