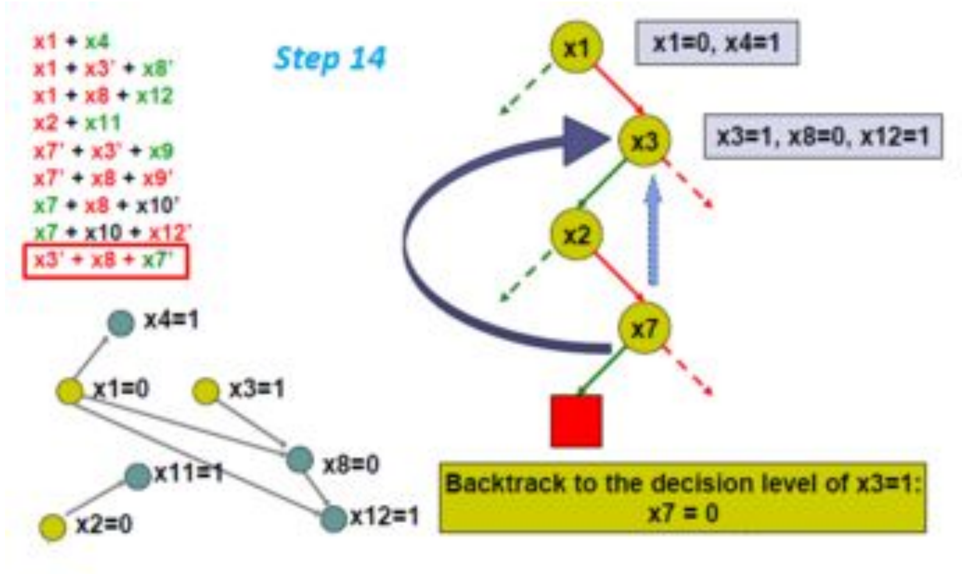
CDCL

Conflict-driven clause learning

What the fork is this shirt?



wiki

Declarative programming

A style of building programs—that expresses the logic of a computation without describing its control flow

generic
constraint
development
environment

Gecode

Regular Expression

Boolean satisfiability problem

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{A}^{\,2} \qquad \overline{B}^{\,1}}{A \land B} \qquad\qquad \dfrac{\overline{A}^{\,2} \qquad \overline{C}^{\,3}}{A \land C}}{(A \land B) \land (A \land C)}\ {}^{1}}{B \to (A \land B) \land (A \land C)}\ {}^{2}}{A \to (B \to (A \land B) \land (A \land C))}\ {}^{3}}{C \to (A \to (B \to (A \land B) \land (A \land C)))}$$

Use case 1

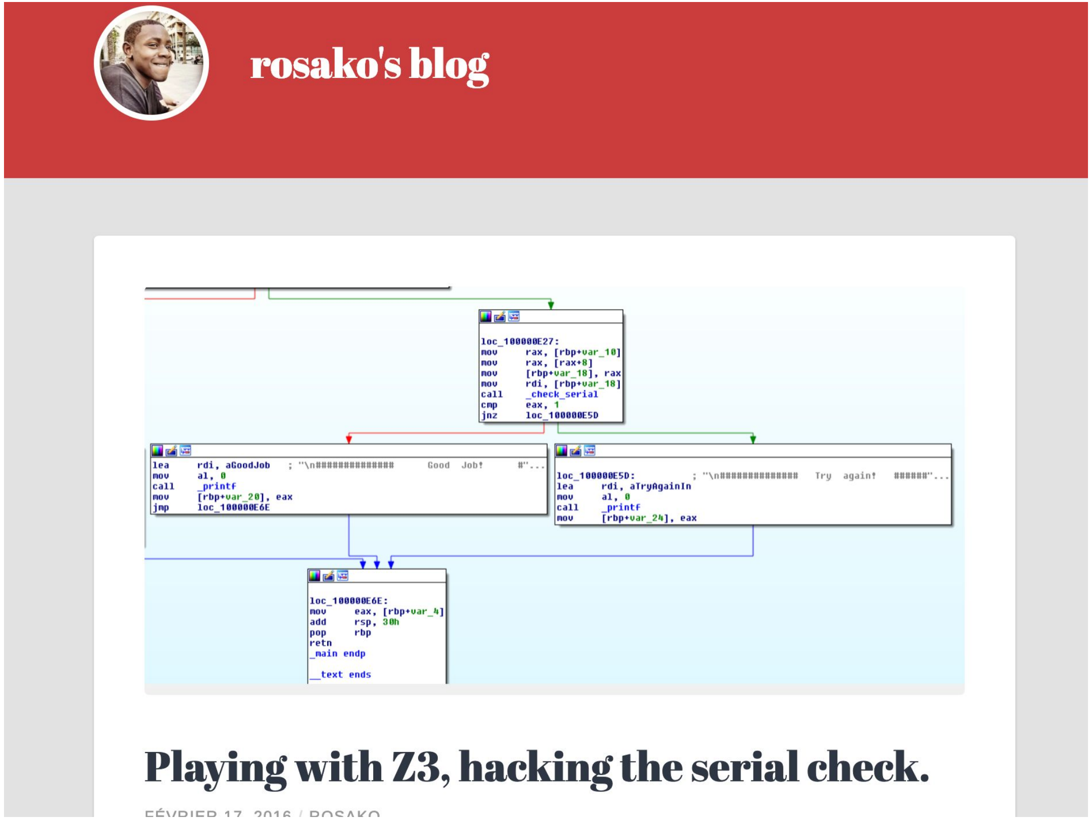
Checking Cloud Contracts in Microsoft Azure

Nikolaj Bjørner[1] and Karthick Jayaraman[2]

[1] Microsoft Research
nbjorner@microsoft.com
[2] Microsoft Azure
karjay@microsoft.com

Abstract. *Cloud Contracts* capture architectural requirements in data-centers. They can be expressed as logical constraints over configurations. Contract violation is indicative of miss-configuration that may only be noticed when networks are attacked or correctly configured devices go off-line. In the context of Microsoft Azure's data-center we develop contracts for (1) network access restrictions, (2) forwarding tables, and (3) BGP policies. They are checked using the SecGuru tool that continuously monitors configurations in Azure. SecGuru is based on the Satisfiability Modulo Theories solver Z3, and uses logical formulas over bit-vectors to model network configurations. SecGuru is an instance of applying technologies, so far developed for program analysis, towards networks. We

Use case 2



rosako's blog
loc_100000E27:
mov rax, [rbp+var_10]
mov rax, [rax+8]
mov [rbp+var_18], rax
mov rdi, [rbp+var_18]
call _check_serial
cmp eax, 1
jnz loc_100000E5D
lea rdi, aGoodJob
mov al, 0
call _printf
mov [rbp+var_20], eax
jmp loc_100000E6E
loc_100000E5D:
lea rdi, aTryAgainIn
mov al, 0
call _printf
mov [rbp+var_24], eax
loc_100000E6E:
mov eax, [rbp+var_4]
add rsp, 30h
pop rbp
retn
_main endp
__text ends
Playing with Z3, hacking the serial check.

Conjunctive normal form

CNF Satisfiability

CNF Boolean Formula

- Variables, $\{x, y, z\}$ in ex.
- Literals, e.g. $x, \bar{y}, \ldots$
- Clauses, disjunctions of literals
- CNF formula, a conjuction of clauses.

A formula is satisfiable if $\exists$ a truth assignment such that the formula evaluates to true.

E.g. $$\underbrace{\underbrace{(\underline{x} \overset{\text{OR}}{\lor} \underline{y})}_{\text{clauses}} \overset{\text{AND}}{\land} \underbrace{(\underline{\bar{y}} \overset{\text{OR}}{\lor} \underline{z})}}_{\text{CNF formula}}$$

literals

in NP

Input: Given a formula.
Decision: Is there a satisfying assignment?

CNF

(A ∨ !B ∨ !C ∨ !D)
∧
(C ∨ !D) ∧
(E)

A = ?
B = ?
C = ?
D = ?
E = ?

CNF

(A ∨ !B ∨ !C ∨ !D)
∧
(C ∨ !D) ∧
(E)

→

A = True
D = False
E = True

DPLL algorithm

Davis–Putnam–Logemann–Loveland (**DPLL**) **algorithm** is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. for solving the CNF-SAT problem.

DPLL algorithm

DPLL(X):

X := unit-resol(X)

if $X = \emptyset$ then return(sat)

if $\perp \notin X$ then

choose variable p in X

DPLL($X \cup \{p\}$)

DPLL($X \cup \{\neg p\}$)

DPLL algorithm

DPLL(X):

$X := \text{unit-resol}(X)$

if $X = \emptyset$ then return(sat)

if $\bot \notin X$ then

  choose variable p in X

  DPLL($X \cup \{p\}$)

  DPLL($X \cup \{\neg p\}$)

DPLL algorithm

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

DPLL algorithm

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

p^d

DPLL algorithm

$\neg p \vee \neg s$ $\quad p \vee r$ $\quad \neg s \vee t$

$\neg p \vee \neg r$ $\quad p \vee s$ $\quad q \vee s$

$\neg q \vee \neg t$ $\quad r \vee t$ $\quad q \vee \neg r$

$p^d \; \neg s$

DPLL algorithm

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

$p^d \ \neg s \ \neg r$

DPLL algorithm

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

$p^d \ \neg s \ \neg r \ t$

DPLL algorithm

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

$p^d \ \neg s \ \neg r \ t \ q$

DPLL algorithm

$\neg p \lor \neg s$	$p \lor r$	$\neg s \lor t$
$\neg p \lor \neg r$	$p \lor s$	$q \lor s$
$\neg q \lor \neg t$	$r \lor t$	$q \lor \neg r$

$p^d \; \neg s \; \neg r \; t \; q$

DPLL algorithm

$\neg p \vee \neg s$ $\quad$ $p \vee r$ $\quad$ $\neg s \vee t$

$\neg p \vee \neg r$ $\quad$ $p \vee s$ $\quad$ $q \vee s$

$\neg q \vee \neg t$ $\quad$ $r \vee t$ $\quad$ $q \vee \neg r$

$p^d \ \neg s \ \neg r \ t \ q$

Backtrack:

$\neg p$

DPLL algorithm

$\neg p \vee \neg s$ $\quad$ $p \vee r$ $\quad$ $\neg s \vee t$

$\neg p \vee \neg r$ $\quad$ $p \vee s$ $\quad$ $q \vee s$

$\neg q \vee \neg t$ $\quad$ $r \vee t$ $\quad$ $q \vee \neg r$

$p^d \ \neg s \ \neg r \ t \ q$

Backtrack:

$\neg p \ r$

DPLL algorithm

$\neg p \lor \neg s$	$p \lor r$	$\neg s \lor t$
$\neg p \lor \neg r$	$p \lor s$	$q \lor s$
$\neg q \lor \neg t$	$r \lor t$	$q \lor \neg r$

$p^d \ \neg s \ \neg r \ t \ q$

Backtrack:

$\neg p \ r \ s$

DPLL algorithm

$\neg p \vee \neg s$	$p \vee r$	$\neg s \vee t$
$\neg p \vee \neg r$	$p \vee s$	$q \vee s$
$\neg q \vee \neg t$	$r \vee t$	$q \vee \neg r$

$p^d \; \neg s \; \neg r \; t \; q$

Backtrack:

$\neg p \; r \; s \; q \; t$

DPLL algorithm

$\neg p \vee \neg s$ $\quad p \vee r$ $\quad \neg s \vee t$

$\neg p \vee \neg r$ $\quad p \vee s$ $\quad q \vee s$

$\neg q \vee \neg t$ $\quad r \vee t$ $\quad q \vee \neg r$

$p^d \ \neg s \ \neg r \ t \ q$

Backtrack:

$\neg p \ r \ s \ q \ t$

Fail

Dimacs

data ▸ simple_v3_c2.cnf.txt

```
c  simple_v3_c2.cnf
c
p cnf 3 2
1 -3 0
2 3 -1 0

```

(X1 ∨ !X3) ∧
(X2 ∨ X3 ∨ !X1)

Python / CNF

```python
def read_dimacs(dimacs_str: str) -> CNF:
```

data ▸ simple_v3_c2.cnf.txt

```
c  simple_v3_c2.cnf
c
p cnf 3 2
1 -3 0
2 3 -1 0

```

```python
[
    {1: True, 3: False},
    {2: True, 3: True, 1: False}
]
```

```python
Clause = Dict[int, bool]
CNF = List[Clause]
```

Python / CNF

```
40  def read_dimacs(dimacs_str: str) -> CNF:
```

```
~/c/cdcl_workshop ❯❯❯ pytest test_before.py test_read_dimacs.py                                   master
================================ test session starts ================================
platform darwin -- Python 3.6.8, pytest-4.3.0, py-1.8.0, pluggy-0.9.0
rootdir: /Users/roman/code/cdcl_workshop, inifile:
plugins: remotedata-0.3.1, pep8-1.0.6, openfiles-0.3.2, cov-2.5.1
collected 2 items

test_before.py .                                                                  [ 50%]
test_read_dimacs.py .                                                             [100%]

============================= 2 passed in 0.03 seconds ==============================
```

Python / State

```python
class Var(NamedTuple):
    name: int
    val: bool
    decision: bool = False
    reason: Optional[Dict[int, bool]] = None

    def __eq__(self, other: Any) -> bool:
        """just ignore reason"""
        if isinstance(other, Var):
            return (self.name == other.name and
                    self.val == other.val and
                    self.decision == other.decision)
        return False


Clause = Dict[int, bool]
CNF = List[Clause]
State = List[Var]
Result = Dict[int, bool]
```

Unit resolution / Reasoning

```python
def unit_resolution_once(cnf: CNF, state: State) -> State:
    """Iterate once over clauses and do unit propagation on CNF

    Args:
        cnf: CNF
        state: new state
    Raises:
        Unsatisfiable: when cannot satisfy CNF
    Returns:
        new updated state
    """
    state_dict = {var.name: var.val for var in state}
    state_vars = set(state_dict.keys())

    for clause in cnf:
        # skip clause if any variables already true
        # TODO

        if no_free_variables:
            raise Unsatisfiable(clause=clause)

        if one_free_variable:
            state.append(one_free_variable)
            return state
    return state
```

Unit resolution / Reasoning

```
~/c/cdcl_workshop ❯❯❯ pytest test_before.py test_read_dimacs.py test_unit_resolution_once.py
============================================================== test session starts =======================
platform darwin -- Python 3.6.8, pytest-4.3.0, py-1.8.0, pluggy-0.9.0
rootdir: /Users/roman/code/cdcl_workshop, inifile:
plugins: remotedata-0.3.1, pep8-1.0.6, openfiles-0.3.2, cov-2.5.1
collected 6 items

test_before.py .
test_read_dimacs.py .
test_unit_resolution_once.py ....

============================================================= 6 passed in 0.03 seconds ====================
```

Unit resolution / Reasoning

```python
def unit_resolution(cnf: CNF, state: State) -> State:
    """Do unit resolution until stuck

    Args:
        cnf: CNF
        state: new state
    Raises:
        Unsatisfiable: when cannot satisfy CNF
    Returns:
        new updated state
    """
    previous_len = len(state)
    while True:
        state = unit_resolution_once(cnf, state)
        if len(state) > previous_len:
            previous_len = len(state)
        else:
            break
    return state
```

Unit resolution / Reasoning

```
~/c/cdcl_workshop ❯❯❯ pytest test_unit_resolution.py                              final
================================= test session starts ==================================
platform darwin -- Python 3.6.8, pytest-4.3.0, py-1.8.0, pluggy-0.9.0
rootdir: /Users/roman/code/cdcl_workshop, inifile:
plugins: remotedata-0.3.1, pep8-1.0.6, openfiles-0.3.2, cov-2.5.1
collected 2 items

test_unit_resolution.py ..                                                       [100%]

=============================== 2 passed in 0.02 seconds ===============================
```

DPLL

```
def cdcl(cnf: CNF, state: Optional[State] = None) -> Result:
    if not state:
        state = []

    while True:
        try:
            state = unit_resolution(cnf, state)
        except Unsatisfiable as e:
            if cannot_backtrack:
                reraise
            else:
                # do backtrack
                # replace_decision_variable_with_no_decision
                continue

        if len(state) == nb_all_variables:
            return {var.name: var.val for var in state}

        # in smart way set some variable
        add_some_random_decision_to_state
```

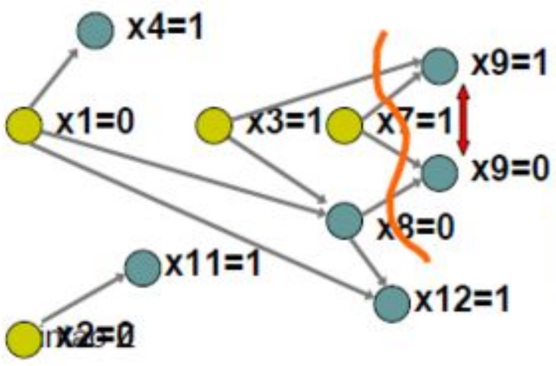
DPLL

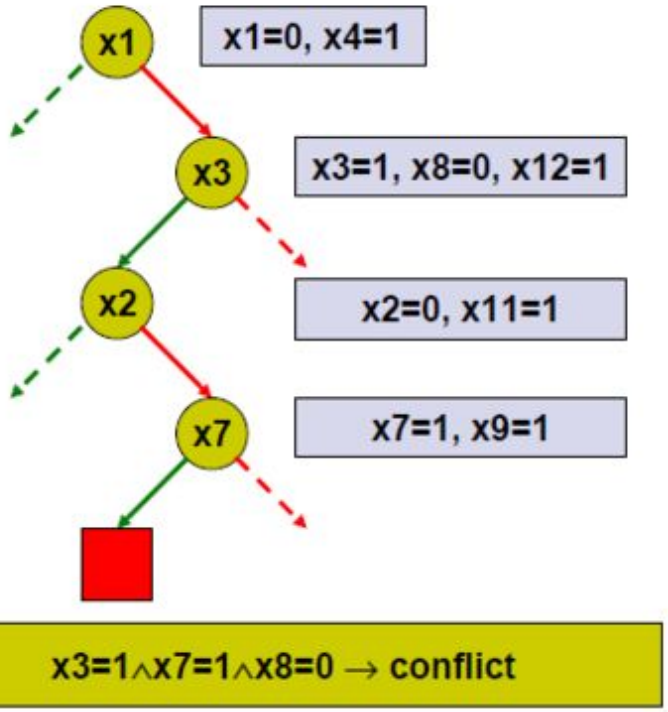
```
~/c/cdcl_workshop ❯❯❯ pytest .                                                    final ✱
================================ test session starts =================================
platform darwin -- Python 3.6.8, pytest-4.3.0, py-1.8.0, pluggy-0.9.0
rootdir: /Users/roman/code/cdcl_workshop, inifile:
plugins: remotedata-0.3.1, pep8-1.0.6, openfiles-0.3.2, cov-2.5.1
collected 10 items

test_before.py .                                                               [ 10%]
test_cdcl.py ..                                                                [ 30%]
test_read_dimacs.py .                                                          [ 40%]
test_unit_resolution.py ..                                                     [ 60%]
test_unit_resolution_once.py ....                                              [100%]

============================= 10 passed in 0.05 seconds ==============================
```

DPLL vs CDCL

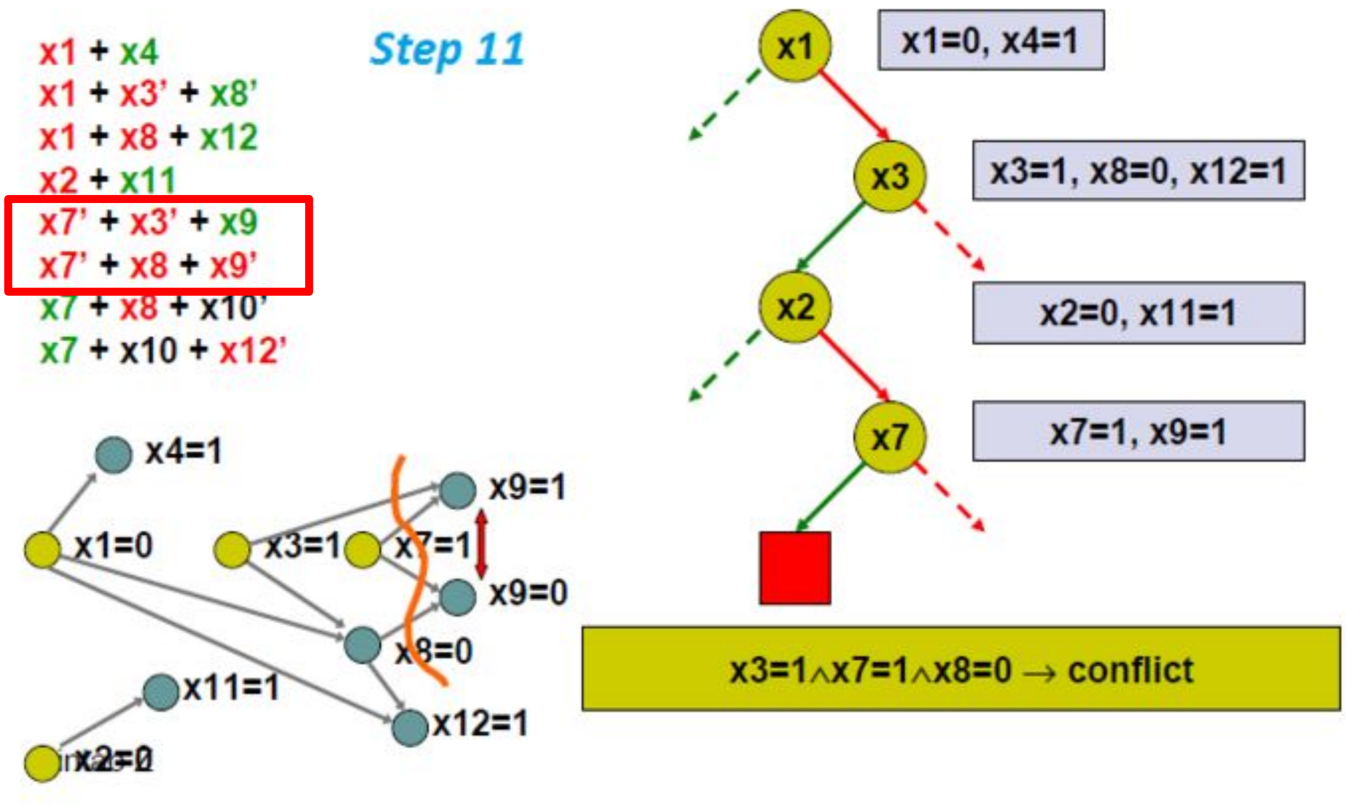
x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'

Step 11





DPLL vs CDCL



x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'
Step 11
x4=1
x1=0
x3=1
x7=1
x9=1
x9=0
x8=0
x11=1
x12=1
x2=0
x1
x3
x2
x7
x1=0, x4=1
x3=1, x8=0, x12=1
x2=0, x11=1
x7=1, x9=1
x3=1∧x7=1∧x8=0 → conflict

DPLL vs CDCL

$x7' + x3' + x9$

$x7' + x8 + x9'$

(!x7 or !x3 or x9) and
(!x7 or x8 or !x9)

→

(!x7 or !x3) or
(!x7 or x8)

→

(!x7 or !x3 or x8)

Thank you!
Questions?