软件分析

约束求解和符号执行

熊英飞
北京大学
2014

路径敏感性

- 路径不敏感分析：不考虑程序中的路径可行性，忽略分支循环语句中的条件
- 路径敏感分析：考虑程序中的路径可行性，只分析可能的路径

路径敏感分析

- 符号执行、模型检查等
- 关键问题：如何知道哪些路径是可行的？
  - 约束求解技术

约束求解

- 给定一组约束，求
  - 这组约束是否可满足
  - 如果可满足，给出一组赋值
  - 如果不可满足，给出最小矛盾集Minimal Unsatisfiable core
- 如
  - a > 10
  - b < 100 || b > 200
  - a+b=30
- 可满足：a=15, b=15

Microsoft Research 微软亚洲研究院

约束求解

- SAT solver：解著名的NP完全问题
- Linear solvers：求线性方程组
- Array solvers: 求解包含数组的约束
- String solver：求解字符串约束

- SMT：综合以上各类约束求解工具

SAT solvers

Slides borrowed from Niklas Een and Sharad Malik

The SAT problem

- A **literal** p is a variable x or its negation $\neg x$.
- A **clause** C is a disjunction of literals: $x_2 \lor \neg x_{41} \lor x_{15}$
- A **CNF** is a conjunction of clauses:
  $(x_2 \lor \neg x_{41} \lor x_{15}) \land (x_6 \lor \neg x_2) \land (x_{31} \lor \neg x_{41} \lor \neg x_6 \lor x_{156})$
- The **SAT-problem** is:
  - Find a boolean assignment
  - such that each clause has a true literal
- First problem shown to be NP-complete (1971)

What's a clause?

A clause of size n can be viewed as n propagation rules:

$$a \vee b \vee c$$

is equivalent to:

$$(\neg a \wedge \neg b) \rightarrow c$$

$$(\neg a \wedge \neg c) \rightarrow b$$

$$(\neg b \wedge \neg c) \rightarrow a$$

Example: Consider the constraint

$t = \text{AND}(x, y)$

x=0 $\rightarrow$ t=0
y=0 $\rightarrow$ t=0
x=1 and y=1 $\rightarrow$ t=1

$\neg x \rightarrow \neg t$
$\neg y \rightarrow \neg t$
$x \wedge y \rightarrow t$

$\{x, \neg t\}$
$\{y, \neg t\}$
$\{\neg x, \neg y, t\}$

$$\neg t \wedge y \rightarrow \neg x$$

Example

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

Unit clause (BCP)

. . .

. . .

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

Another unit clause (more BCP)

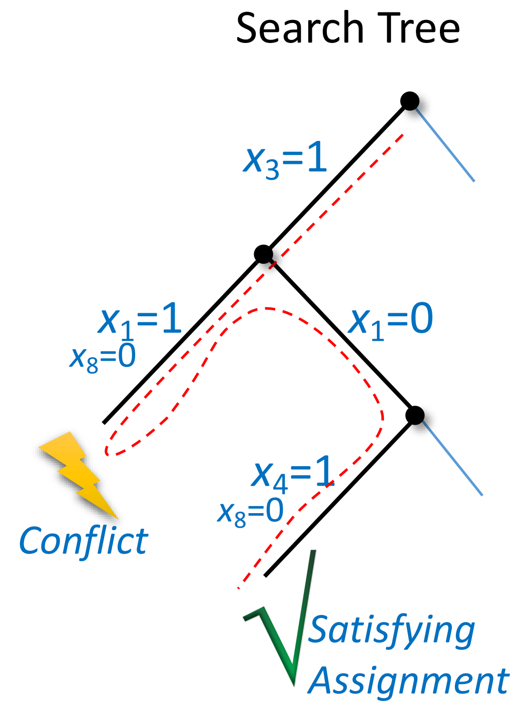
{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

CONFLICT! (backtrack)

Search Components

- Decision heuristic
  - Static (x_1, x_2, x_3...)
  - State based
    - Shortest non-satisfied clause, most common literal etc.
  - History based
    - Pick variables that lead to conflicts in the past.
- Propagation
- Backtracking

Decision heuristic
Propagation
Backtracking



Microsoft Research 微软亚洲研究院

Search Components

- Decision heuristic
- Propagation
  - Unit propagation ("BCP, Boolean Constraint Propagation")
  - Unate propagation
  - Probing/Dilemma
  - Equivalence classes
- Backtracking

~~{**3**, 6, -7, 8}~~

{1, 4, 7}

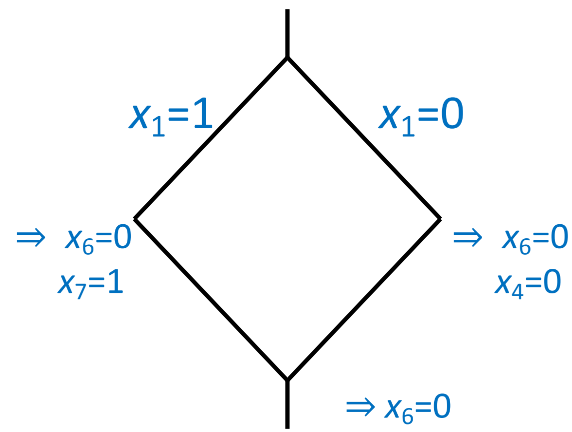
{-8, 4}

Unate (pure literal)

{-1, **-3**, 8}

{**-3**, -4, -8}

~~{-1, -2, **3**, 4, -6}~~

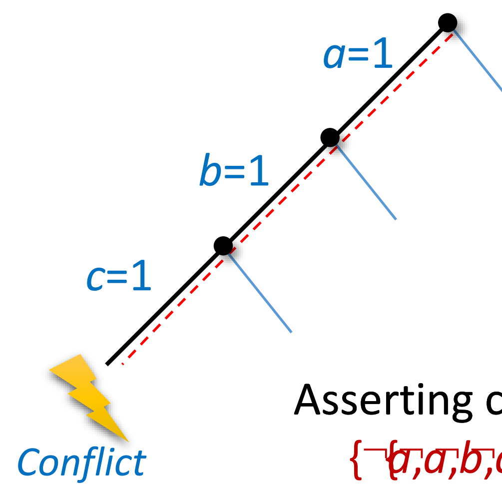


Search Components

- Decision heuristic
- Propagation
- Backtracking
  - Flip last decision
    (standard recursive backtracking)
  - Conflict analysis:
    - Learn an ***asserting clause***
    - [...]

```
dpll(assign){
   "do BCP";
   if "conflict": return FALSE;
   if "complete assign": return TRUE;
   "pick decision variable x";
   return dpll(assign[x=0])
          || dpll(assign[x=1]);
}
```

- May be expressed in any variables, not just decisions.
- Must have only *one* variable from the last decision level.



Asserting clause:
$\{\neg a, \neg b, \neg c\}$

What if b was irrelevant?

Search Components

- Decision heuristic
- Propagation
- Backtracking
  - Flip last decision
    (standard recursive backtracking)
  - Conflict analysis:
    - Learn an asserting clause
    - Backjumping
    - No recursion
    - Can be viewed as a resolution strategy, guided by conflicts.
    - Together with *variable activity* most important innovation.
    - CDCL=Conflict-Driven Clause Learning

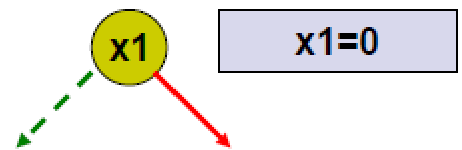
```
dpll(assign){
   "do BCP";
   if "conflict": return FALSE;
   if "complete assign": return TRUE;
   "pick decision variable x";
   return dpll(assign[x=0])
          || dpll(assign[x=1]);
}
```

```
forever{                          – CDCL procedure
  "do BCP"
  if "no conflict":
     if "complete assign": return TRUE;
     "pick decision x=0 or x=1";
  else:
     if "at top-level": return FALSE;
     "analyze conflict"
     "undo assignments"
     "add conflict clause"
}
```

An example

Step 1

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

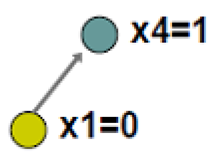


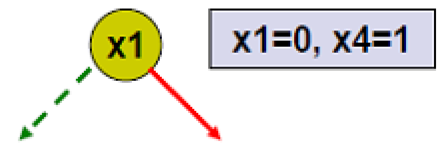
x1=0

An example

Step 2

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

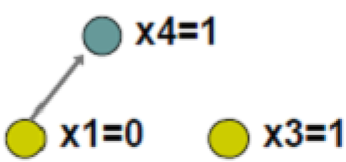


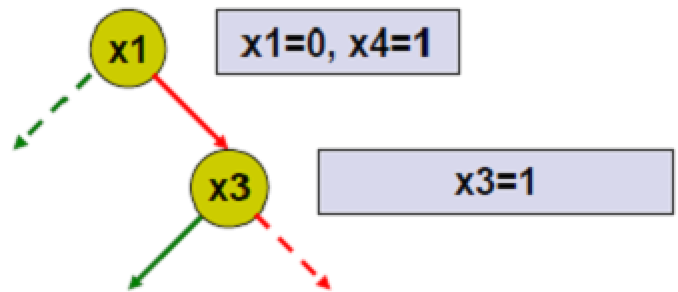


An example

Step 3

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

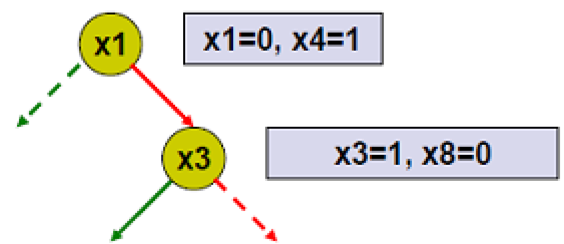


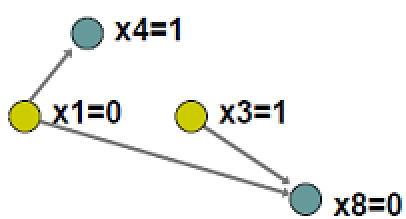


An example

Step 4

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

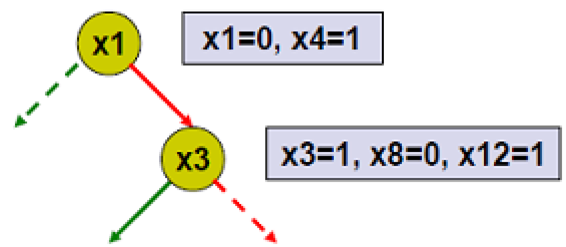


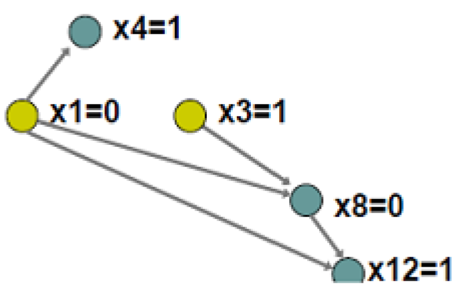


An example

Step 5

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

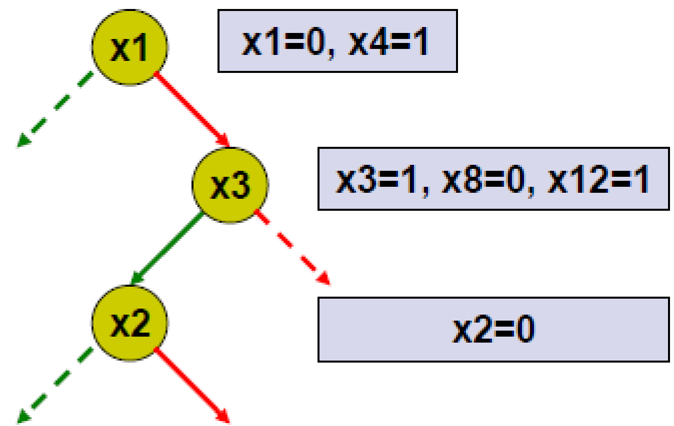


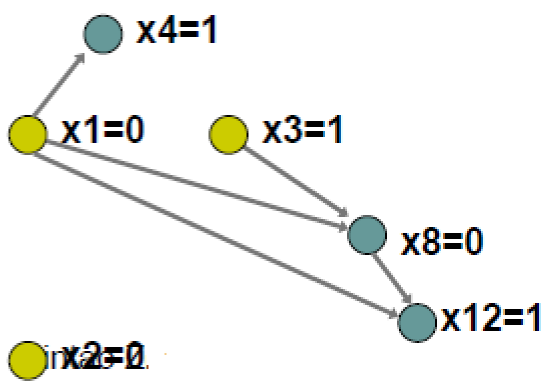


An example

Step 6

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

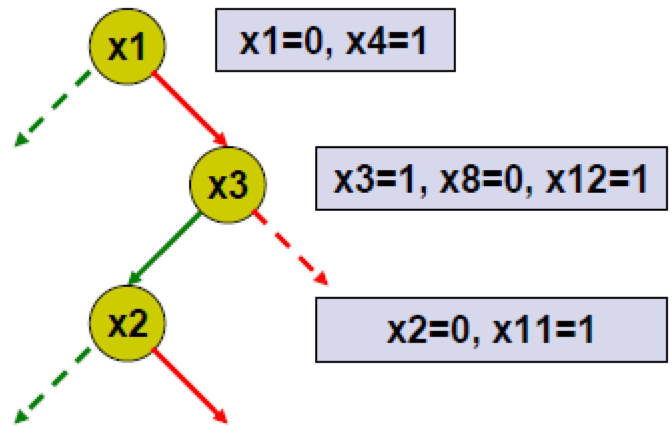


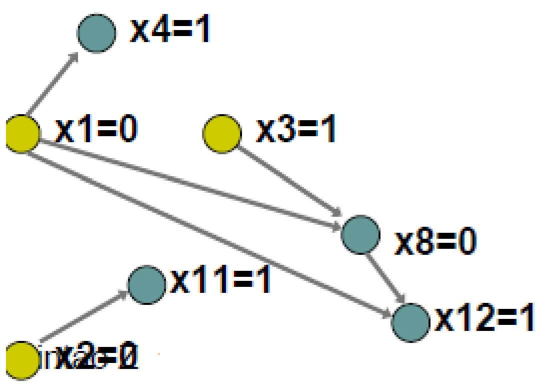


An example

Step 7

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

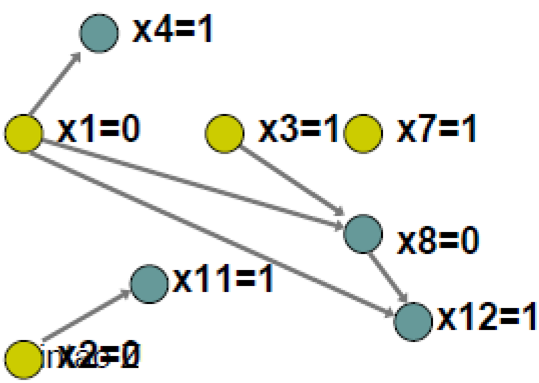


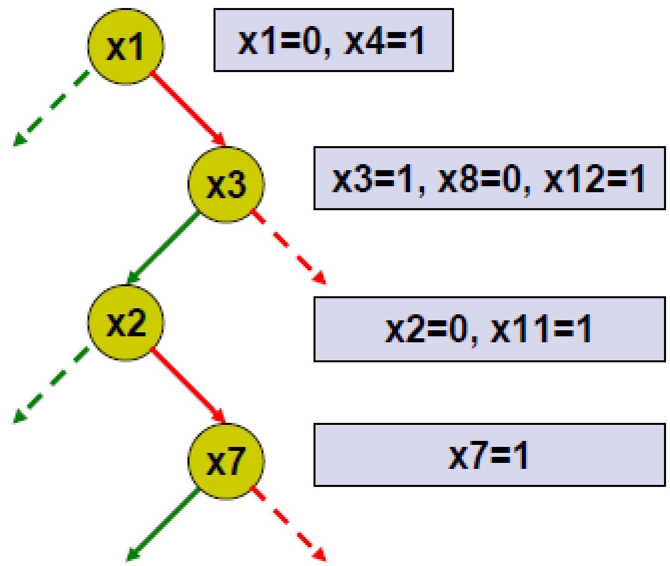


An example

Step 9

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

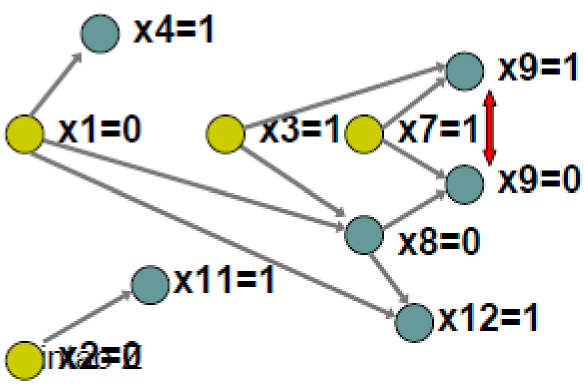


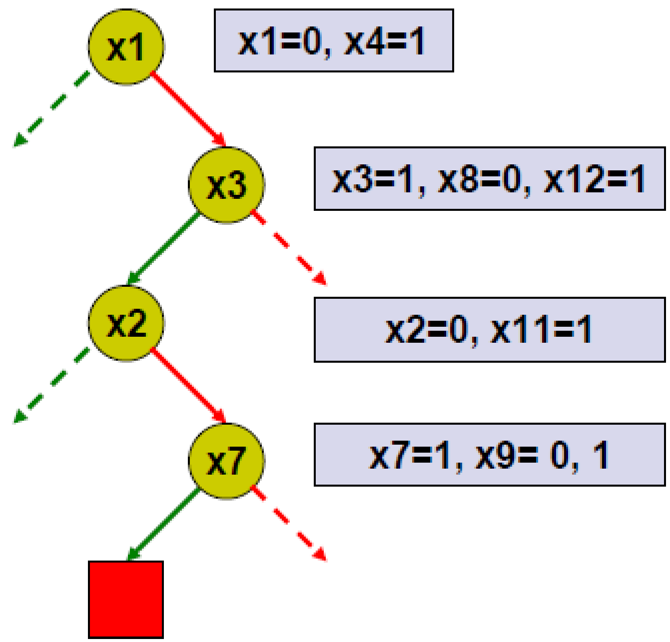


An example

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

Step 10

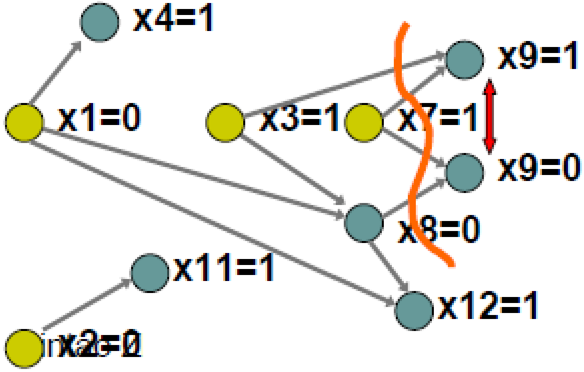


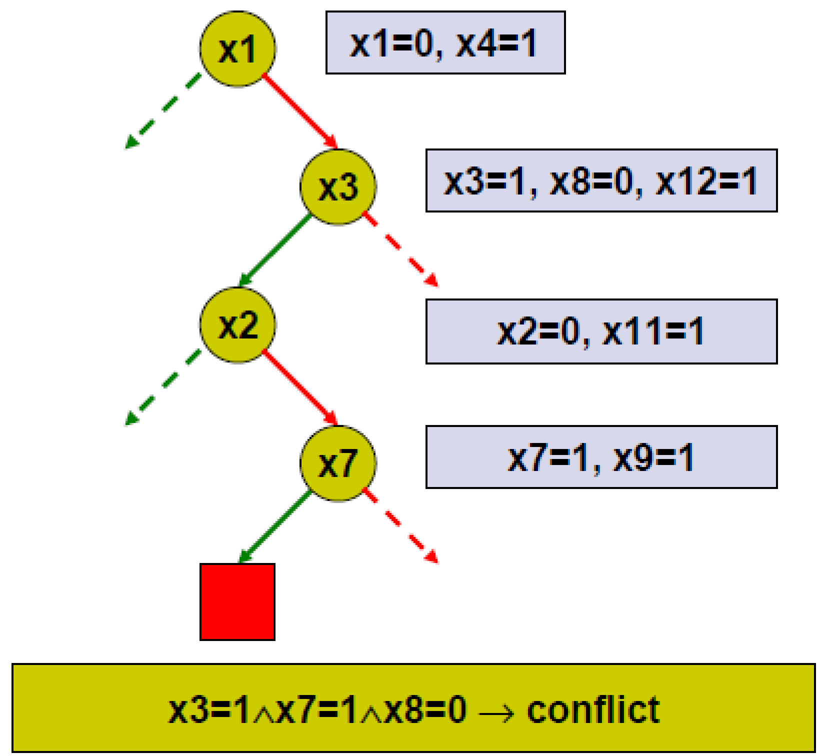


An example

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

Step 11





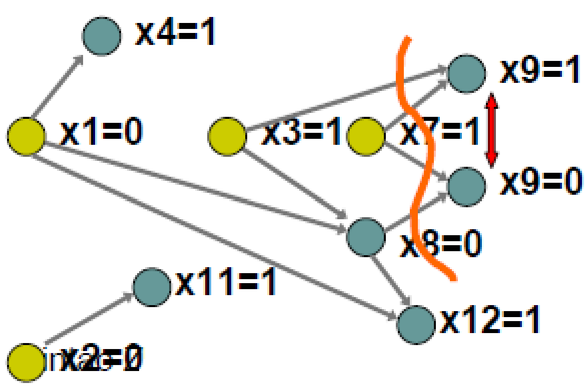
An example

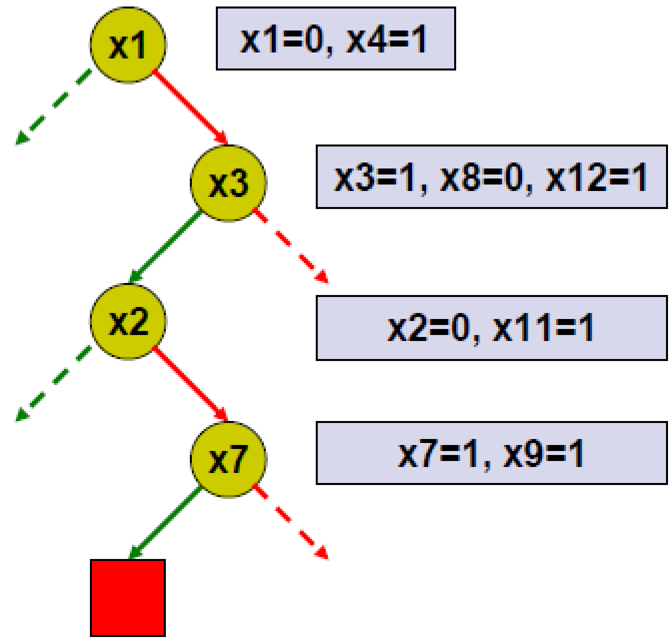
- 化简
  - x3=1/\x7=1/\x8=0 -> false
- 得到
  - x3’+x7’+x8

An example

Step 13

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



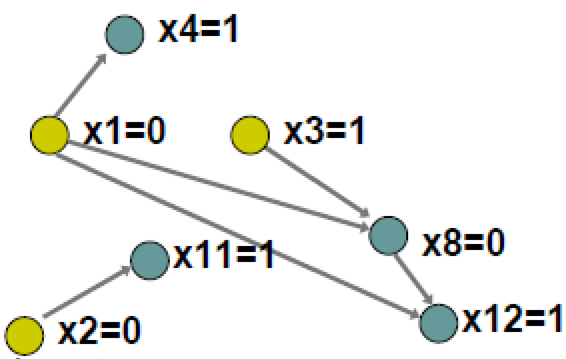


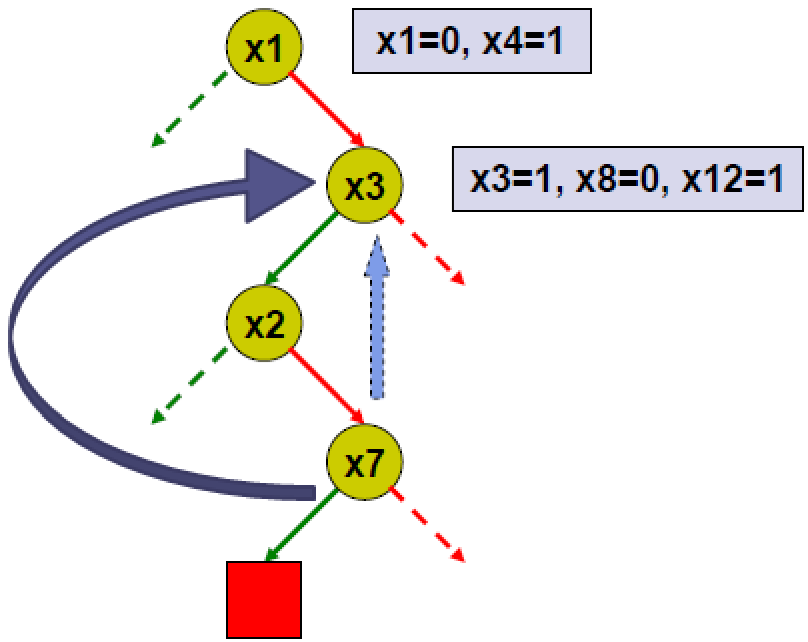
x3=1 $\wedge$ x7=1 $\wedge$ x8=0 $\rightarrow$ conflict
Add conflict clause: x3'+x7'+x8

An example

x1 + x4
x1 + x3' + x8'
x1 + x8 + x12
x2 + x11
x7' + x3' + x9
x7' + x8 + x9'
x7 + x8 + x10'
x7 + x10 + x12'
$\boxed{\text{x3' + x8 + x7'}}$

Step 14



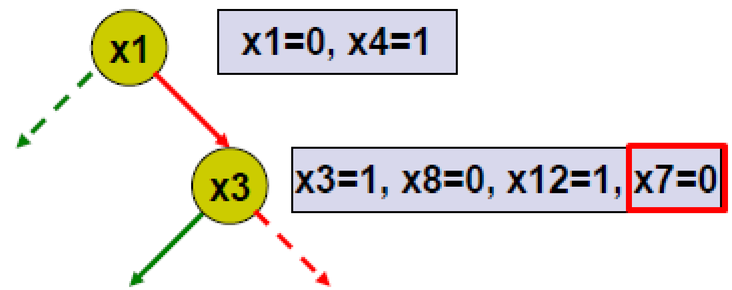


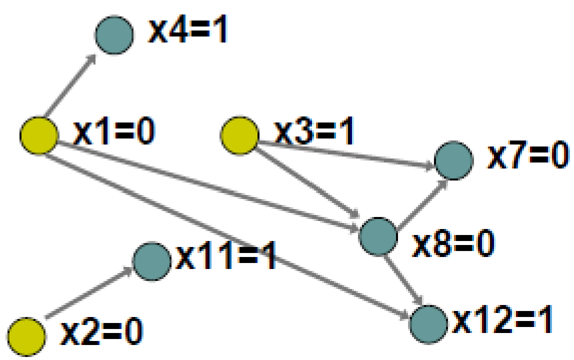
Backtrack to the decision level of x3=1:
x7 = 0

An example

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

Step 15





Variable Activity

- The VSIDS activity heuristic:
  - Rank variables by literal count in the initial clause database
  - Only increment counts as new clauses are added.
  - Periodically, divide all counts by a constant

SMT Solver

using the slides from Albert Oliveras

SMT Solver的使用

- SMT-LIB
  - 标准的SMT输入格式
  - 被几乎所有的SMT Solver支持
  - 用于每年的SMT比赛中

SMT-LIB by Example

- > (declare-fun x () Int)
- > (declare-fun y () Int)
- > (assert (= (+ x (* 2 y)) 20))
- > (assert (= (- x y) 2))
- > (check-sat)
- sat
- > (get-value (x y))
- ((x 8)(y 6))
- > (exit)

Scope

- > (declare-fun x () Int)
- > (declare-fun y () Int)
- > (assert (= (+ x (* 2 y)) 20))
- > (push 1)
- > (assert (= (- x y) 2))
- > (check-sat)
- sat
- > (pop 1)
- > (push 1)
- > (assert (= (- x y) 3))
- > (check-sat)
- unsat
- > (pop 1)
- > (exit)

Defining a new type

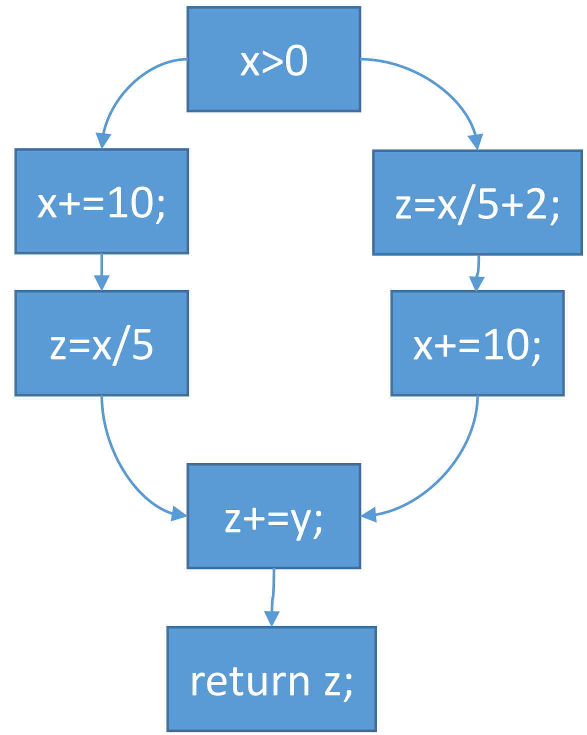
- > (declare-sort A 0)
- > (declare-fun a () A)
- > (declare-fun b () A)
- > (declare-fun c () A)
- > (declare-fun d () A)
- > (declare-fun e () A)
- > (assert (or (= c a)(= c b)))
- > (assert (or (= d a)(= d b)))
- > (assert (or (= e a)(= e b)))
- > (push 1)
- > (distinct c d)
- > (check-sat)
- sat
- > (pop 1)
- > (push 1)
- > (distinct c d e)
- > (check-sat)
- unsat
- > (pop 1)
- > (exit)

常见的SMT Solver

- Z3
  - 微软开发
  - 目前使用最广稳定性最好
  - 仅支持Windows，不开源
- Yices 2
  - Z3之前使用最广稳定性最好的Solver
  - 由Z3的作者在加入微软之前撰写
  - 支持所有平台，开源

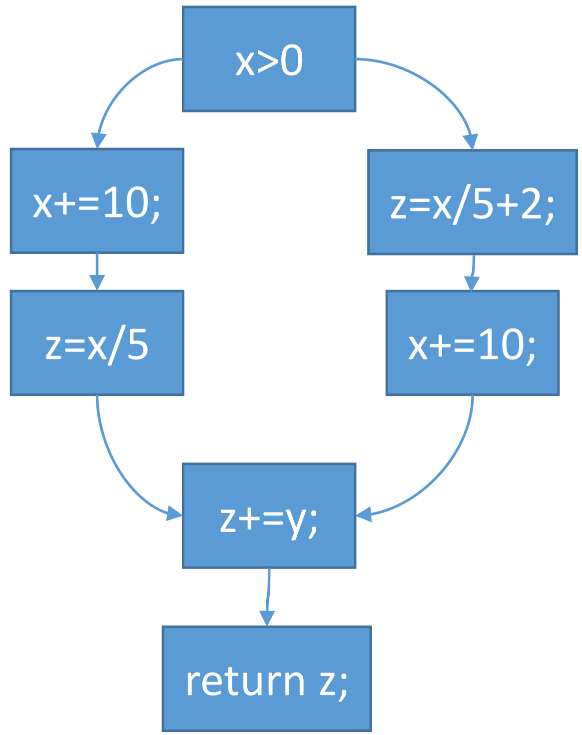
符号执行

```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



符号执行

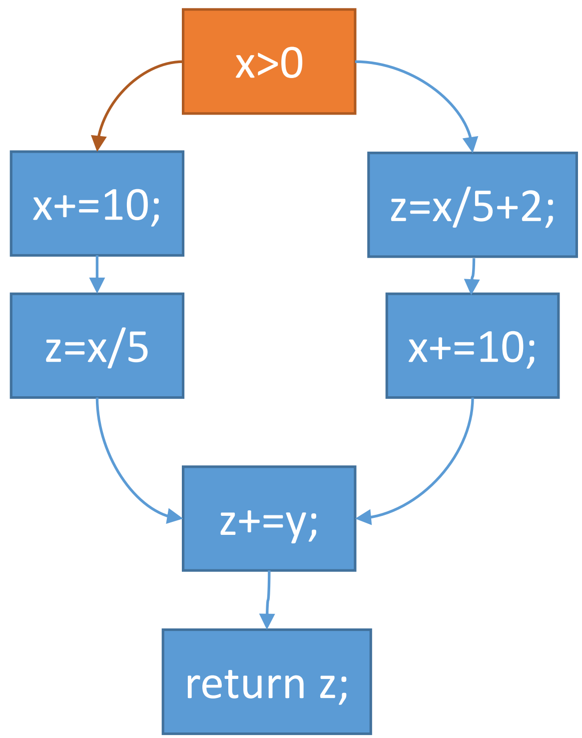
- int main(x,y) {
-   if (x>0) {
-     x+=10;
-     z=x/5;
-   }
-   else {
-     z=x/5+2;
-     x+=10;
-   }
-   z+=y;
-   return z;
- }



x=a
y=b
z=?

符号执行

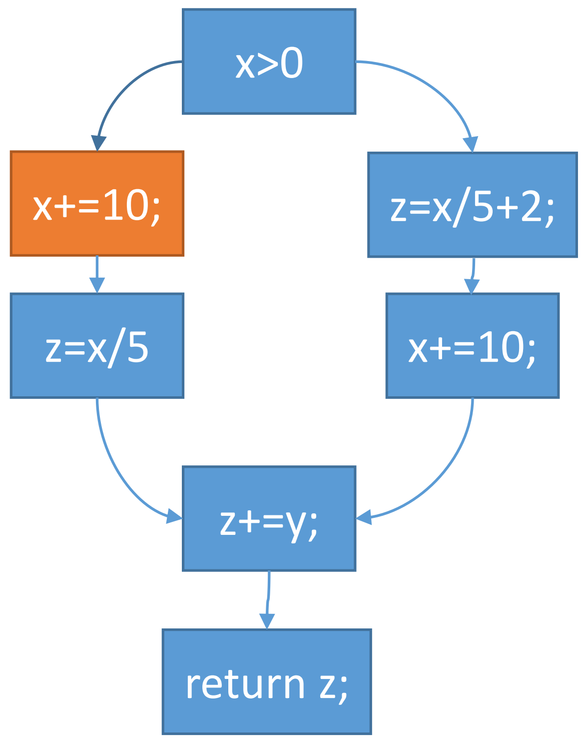
- int main(x,y) {
- if (x>0) {
- x+=10;
- z=x/5;
- }
- else {
- z=x/5+2;
- x+=10;
- }
- z+=y;
- return z;
- }



x=a
y=b
z=?
a>0

符号执行

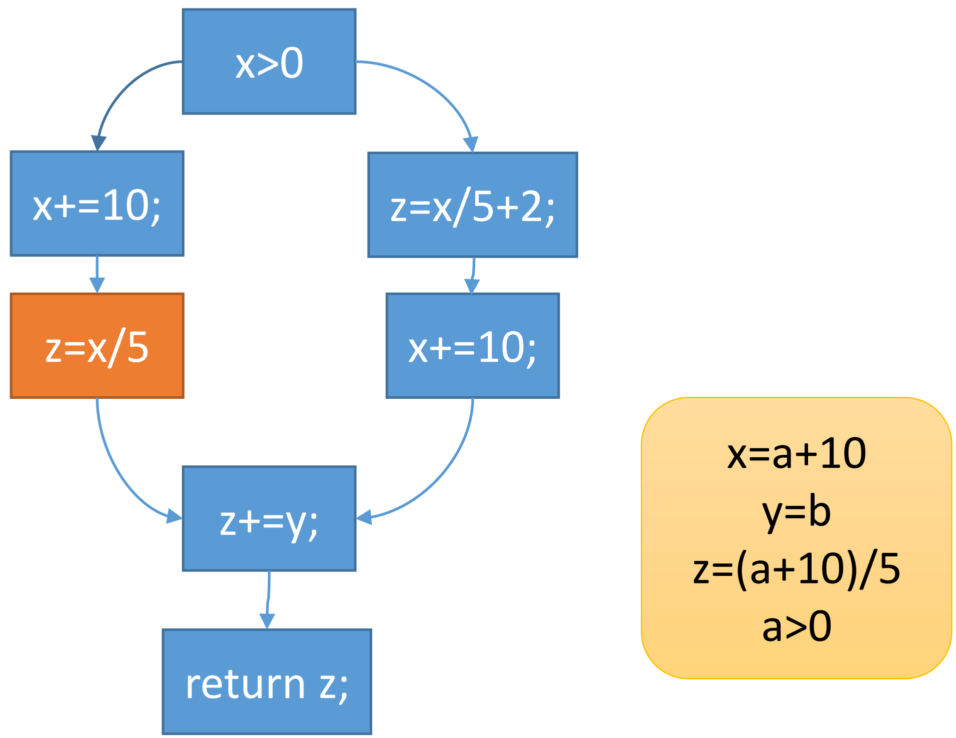
- int main(x,y) {
- if (x>0) {
- x+=10;
- z=x/5;
- }
- else {
- z=x/5+2;
- x+=10;
- }
- z+=y;
- return z;
- }



x=a+10
y=b
z=?
a>0

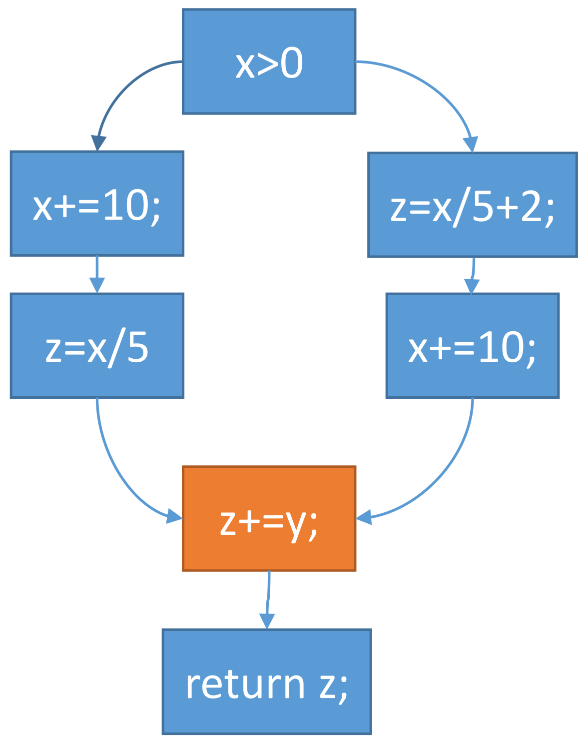
符号执行

- int main(x,y) {
- if (x>0) {
- x+=10;
- z=x/5;
- }
- else {
- z=x/5+2;
- x+=10;
- }
- z+=y;
- return z;
- }



符号执行

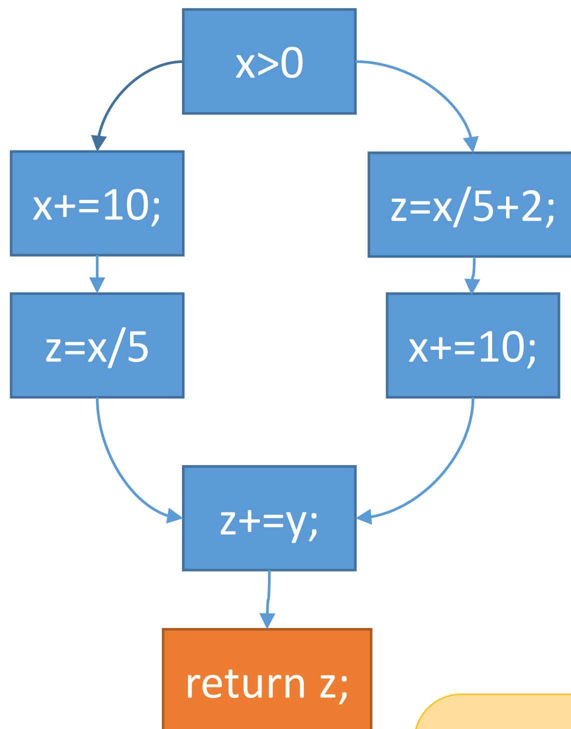
- int main(x,y) {
- if (x>0) {
- x+=10;
- z=x/5;
- }
- else {
- z=x/5+2;
- x+=10;
- }
- z+=y;
- return z;
- }



x=a+10
y=b
z=(a+10)/5+b
a>0

符号执行

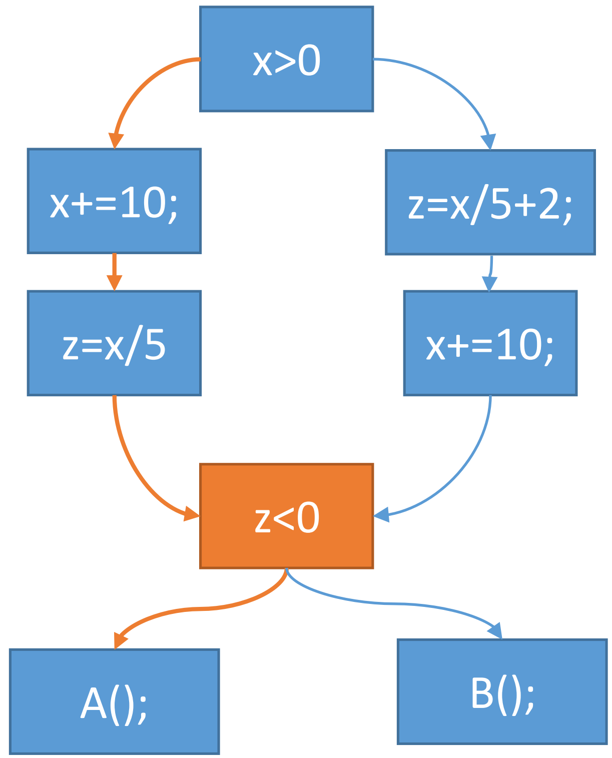
```
int main(x,y) {
  if (x>0) {
    x+=10;
    z=x/5;
  }
  else {
    z=x/5+2;
    x+=10;
  }
  z+=y;
  return z;
}
```



x=a+10
y=b
z=(a+10)/5+b
a>0

返回值为(a+10)/5+b
且a>0

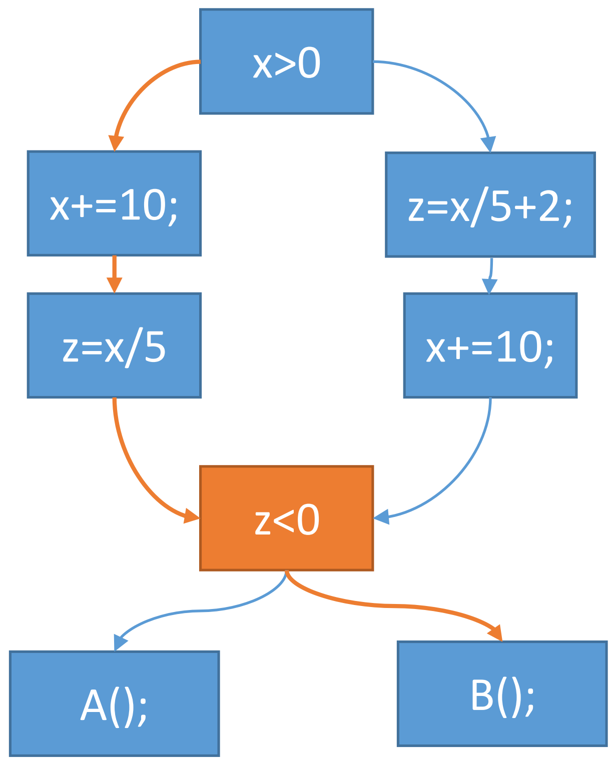
路径可行性



x=a+10
y=b
z=(a+10)/5
a>0

a>0/\ (a+10)/5<0不可满足

路径可行性



x=a+10
y=b
z=(a+10)/5
a>0

a>0/\ (a+10)/5<0不可满足

符号执行

- 程序的规约通常表示为前条件和后条件
  - 前条件：a>0, b>0
  - 后条件：return > 0
- 形成命题：
  - (a+10)/5+b > 0 /\ a>0 /\ b>0
  - 命题成立=逆命题不可满足
  - 用SMT Solver可求解
- 规约被违反=任意路径对应的命题不成立
- 规范被满足=所有路径对应的命题都成立
  - 通常做不到
  - 对于循环，遍历有限次

课后作业

- 下载安装任意SMT Solver
- 发邮件给助教，回答如下问题：
  - 该SMT Solver的名字
  - 该SMT Solver支持的Theory
  - 构造该SMT Solver无法求解的约束，将运行结果截屏附在邮件中
  - 解释该SMT Solver为什么不能求解这个约束