Exercise 03

2021/10/29

Supplemental Material

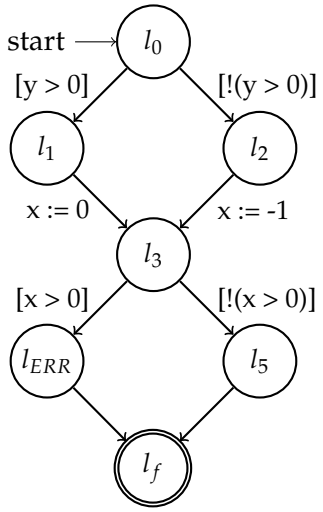
In this sheet we will cover concepts from section 2.1 (Program Representation) of A Unifying View on SMT-Based Software Verification, namely the strongest post computation, that we will use to construct a Bounded Model Checking (BMC) analysis that can be seen as a single edge case of the more general CPA described in section 3.1 of A Unifying View on SMT-Based Software Verification. (As additional reading, please consider the original paper on SAT-based Bounded Model Checking: Symbolic model checking without BDDs. You can leave out sections 3-5 since the scope and implementation details of their approach vary from ours.)

Goal

This time we will concentrate on analyses that make use of first-order logical formulas that can then be solved by making use of BDDs or with a SAT or an SMT solver. The semantics of a program path can be expressed by such a formula, which will lead to a straight-forward way of model checking.
In exercises 1 and 2 we will learn how to construct these formulas and how to get rid of quantifiers in them. Exercise 3 will then show how this can be used to construct a simple yet precise model checking algorithm.
Exercise 4 will contain some practical programming tasks that involve querying an SMT solver.

```
1  int x;
2  int y; // defined, but unknown
3
4  if (y > 0) {
5  x = 0;
6  } else {
7  x = -1;
8  }
9
10 if (x > 0) {
11 ERR:;
12 }
```

Program 1



CFA $A = (L, E, l_0)$

Figure 1: Example program from the last exercise sheet

1 Strongest Postcondition (synchronous)

This exercise is designed to practice the concepts explained in section 2.1 (Program Representation) of A Unifying View on SMT-Based Software Verification.

For adequately capturing the semantics of an operation in a program we can make use of the strongest post operator SP. Remember that in our simple programming language we model every operation as either an assumption or an assignment, so we need to consider these two cases:

- Assume [expr]: $SP([expr], f) = f \wedge expr$
- Assignment v=expr;: $SP(v = expr;, f) = \exists v' : f[v \to v'] \wedge v = expr[v \to v']$

The complicated case here is the assignment, where we have to rename the variables such that v still describes the latest value of that variable. The old value will be renamed to v' and existentially quantified.

Of course this notion can easily be extended to sequences of operations such that we can create formulas for program paths (which we will often call *path formulas* then):

- $SP(\varepsilon, f) = f$
- $SP((op_1, \ldots, op_n), f) = SP((op_2, \ldots, op_n), SP(op_1, f))$

Now for some practice, consider following sequences of operations G_i. Compute strongest post condition $SP(G_i, true)$ and determine which formulae are satisfiable.

a) $G_1 : ([x > 0], y = 1;, [x <= 0])$

b) $G_2 : ([x > 0], y = 2;, [x > 0])$

c) $G_3 : ([!(x > 0)], y = -1;, [y == 0])$

d) $G_4 : (y = 0;, [x > 0], y = 5;, [x > 10])$

2 Skolemization via Single Static Assignment (SSA)

(synchronous, but can also be done asynchronously if short on time)

This exercise is designed to practice the concepts explained in section 2.1 (Program Representation) of A Unifying View on SMT-Based Software Verification.

The formulas created in the last exercise have the disadvantage that they potentially contain many existential quantifiers. With all these quantified variables, it is also sometimes hard to figure out what is going on and in which order things were executed.

A way to address this is using single static assignment, where a new identifier is introduced every time a variable changes. This way we can enforce that each identifier is only assigned once. This form of encoding is also very popular in compilers, as it makes it easier to reason about the program.

A new identifier is usually created by indexing the variable name with a increasing number. This will become clear with an example: $SP((x = x + 1;, x > 0, x = x + y;), x_0 = 0)$ can be expressed as $x_0 = 0 \wedge x_1 = x_0 + 1 \wedge x_1 > 0 \wedge x_2 = x_1 + y_0$ using the SSA form.

Express the following strongest post conditions $SP(G_i, true)$ using the SSA form and determine which formulae are satisfiable.

a) $G_1 : ([x > 0], y = 1;, [x <= 0])$

b) $G_2 : ([x > 0], y = 2;, [x > 0])$

c) $G_3 : ([!(x > 0)], y = -1;, [y == 0])$

d) $G_4 : (y = 0;, x > 0, y = 5;, x > 10)$

3 Bounded Model Checking (asynchronous)

This exercise is designed to practice bounded model checking, which is a simple variant of the CPA explained in section 3.1 of A Unifying View on SMT-Based Software Verification.
The strongest postcondition can be used to construct a very simple model checking analysis. Usually this is done as part of bounded model checking, where all paths of a program up to a certain (bounded) length are checked.

For this we can introduce the configurable program analysis $\mathbb{F}$, which consists of:

1. the abstract domain $D_{\mathbb{F}} = (C, \mathcal{E}, [\![\cdot]\!])$ consists of the set C of concrete states, the lattice $\mathcal{E}$, and the concretization function $[\![\cdot]\!]$. The semi-lattice $\mathcal{E} = (\mathcal{F}, \sqsubseteq_{\mathbb{F}}, \sqcup_{\mathbb{F}}, \top)$ is given by the set $\mathcal{F}$ of all formulas over the program variables, the subsumption relation $\sqsubseteq_{\mathbb{F}}$ that checks entailment $\phi \sqsubseteq_{\mathbb{F}} \psi \Leftrightarrow (\phi \Rightarrow \psi)$ and $\top$ is the tautology, i.e., the formula that is always true. The concretization function $[\![\cdot]\!]$ is given by $[\![\phi]\!] = \{c | c \in C \wedge c \models \phi\}$.

   Note that there might be syntactically different yet semantically identical formulas that are considered to be equal in this definition! We more or less use formulas here to talk about subsets of C, so we consider two formulas to be equivalent if $[\![\phi]\!] = [\![\psi]\!]$, which is the same as $\phi \Leftrightarrow \psi$.

2. the transfer relation $\rightsquigarrow_{\mathbb{F}}$ contains $\phi \overset{g}{\rightsquigarrow} \phi'$ if for $g = (l, op, l')$ it holds that $SP(op, \phi) = \phi'$. Equality here means that it is syntactically the formula that is created by the strongest post (to avoid having to deal with multiple formulas as successor that are semantically equal).

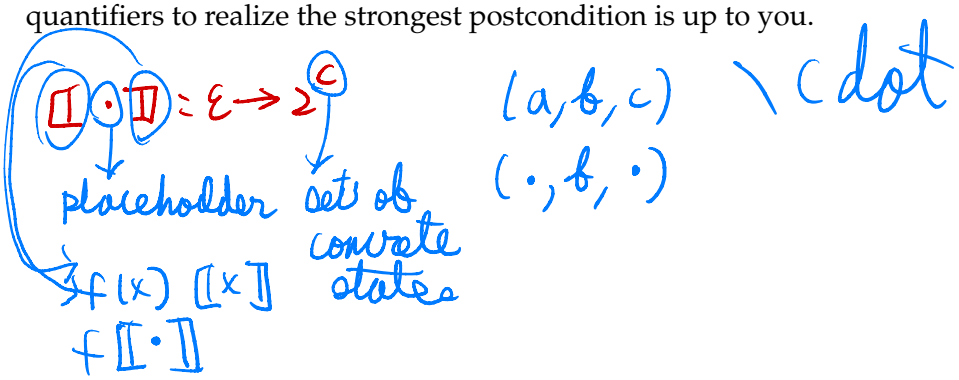
   We can additionally demand that ϕ' must be satisfiable, which will require a SAT check. In practice this SAT check is often expensive so it we could e.g. only do it error/target states, but for simplicity we assume here that the transfer relation only contains $\phi \overset{g}{\rightsquigarrow} \phi'$ if ϕ' is satisfiable.

3. the merge operator is defined by $\texttt{merge}_{\mathbb{F}}(\phi, \psi) = \psi$, i.e., we never merge

4. the stop operator is defined by $\texttt{stop}_{\mathbb{F}} = false$, i.e., we never stop

With this knowledge, we can now put what we learned in exercise 1 and 2 into practice:

a) Run the CPA algorithm with the configurable program analysis $\mathbb{L} \times \mathbb{F}$ on the program in Fig. 1. Use $e_0 = (l_0, \top)$ as the initial state. Whether you use SSA form or existential quantifiers to realize the strongest postcondition is up to you.



CPA $\mathbb{F} = (D, \rightsquigarrow^g, merge, stop)$

transfer relation

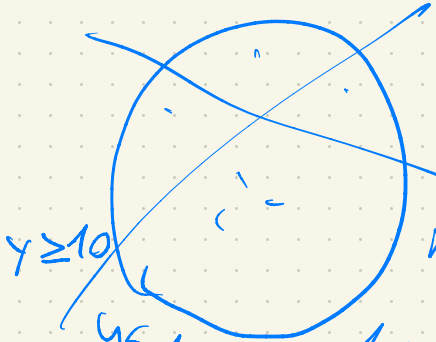
$T_{IB} \rightsquigarrow^g T_{IB}$ $e_0 = (l_0, T, 10) \rightsquigarrow^g (l_1, x=0, y)$

$\mathbb{L} \times \mathbb{F} \times \mathbb{B}$

composit CPA

$g = (l_0, x:=0, l_1)$

$T \rightsquigarrow^g x=0$ like $x=1$ $y=2$ ....



bound

yes ⇒ bus

no ⇒ unknown

how to get rid of "bounded"

how to partition the state space

in (few) subsets ?

$\varphi \equiv y \geq 10 \wedge x$

BMC $\bigvee_{0 \leq i \leq n} \varphi$ SAT ?

Now this works well as long as there are no loops in the program. With loops like in Fig. 2 however, there is no guarantee that our analysis will ever terminate. For practical reasons we might therefore want to limit the exploration to a certain maximal path length. We can achieve this with the Bounds CPA $\mathbb{B}$, which will look as follows:

1. the abstract domain $D_{\mathbb{B}} = (C, \mathcal{E}, [\![\cdot]\!])$ consists of the set C of concrete states, the lattice $\mathcal{E}$, and the concretization function $[\![\cdot]\!]$. The semi-lattice $\mathcal{E} = (\mathbb{N} \cup \{\top_{\mathbb{B}}\}, \sqsubseteq_{\mathbb{B}}, \sqcup_{\mathbb{B}}, \top_{\mathbb{B}})$ is a flat semi-lattice where the elements are from the set $\mathbb{N}$ of non-negative integers. We add a top element $\top_{\mathbb{B}}$ for formal reasons only and will actually never make use of it. The concretization function $[\![\cdot]\!]$ is given by $[\![a]\!] = C$.

2. the transfer relation $\rightsquigarrow_{\mathbb{B}}$ contains $a \overset{g}{\rightsquigarrow} b'$ if $b = a - 1$ and $b \geq 0$. Note that the second condition $b \geq 0$ will enable us to stop exploring once we have counted down to 0, and that this behavior violated the soundness criterion for transfer relations you learned about in the last sheet.

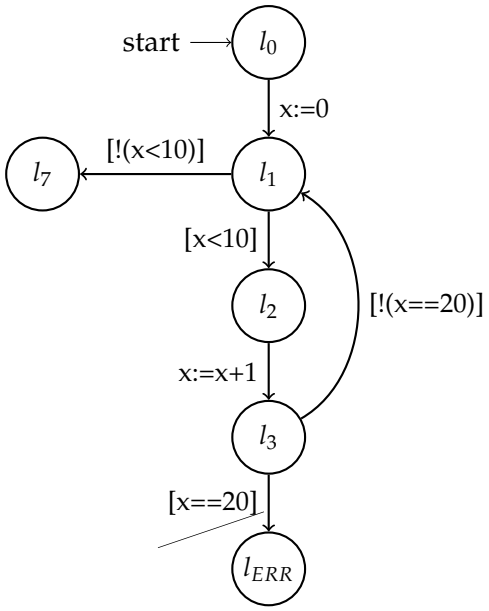
3. the merge operator is defined by $\texttt{merge}_{\mathbb{B}}(a, b) = b$, i.e., we never merge

4. the stop operator is defined by $\texttt{stop}_{\mathbb{F}} = \texttt{stop}^{sep}$

b) Run the CPA algorithm with the configurable program analysis $\mathbb{L} \times \mathbb{F} \times \mathbb{B}$ on the program in Fig. 2. Use $e_0 = (l_0, \top, 10)$ as the initial state. Whether you use SSA form or existential quantifiers to realize the strongest postcondition is up to you.

```
0  int x = 0;
1  while (x < 10) {
2    x = x + 1 ;
3    if (x == 20) {
4      ERR:;
5    }
6  }
7
```

Program 2



CFA $A = (L, E, l_0)$

Figure 2: A program with a loop, taken from the third exercise sheet

4 Programming Exercise on BMC Using SMT Solvers

Go to the lecture's Jupyter notebook repository [1] and complete the programming tasks in the Jupyter Notebook named `03_BMC.ipynb`.

[1] https://gitlab2.cip.ifi.lmu.de/sosy-lab/sv-ws-20-notebooks

$\Psi_4 := SP(y := 0;, [x>0], y=5;, [x>10]),$ true)

$= SP((\,[x>0], y := 5;, [x>10]), \exists y' : \underbrace{\text{true} \wedge y=0}_{y=0}$

$= SP((y := 5;, [x>10]),\ y=0 \wedge x>0)$

bound variable

$= SP([x>10], \exists y' : y'=0 \wedge x>0 \wedge y=5)$

$(\exists y' : y'=0) \wedge x>0 \wedge y=5$

true

quantifier (invalid)

verbose

$= x>0 \wedge y=5 \wedge x>10$

SAT ex: $y=5$, $x=11$

$SP((x:=3;, [y>0], x=2x+1, true)$

$= SP([y>0], x:=2x+1;), x=3)$

$= SP(x:=2x+1;, x=3 \wedge y>0)$

$= \exists x' : x'=3 \wedge y>0 \wedge x = 2x'+1$

$(x'=3) \; x=7 \wedge y>0$ SAT (satisfiable)

ex: $x=7$, $y=1$ #

$G_2 = SP([x>0], y=2;, [x>0]), true)$

$= SP(y=2;, [x>0]), SP([x>0], true)$

$= SP(y=2;, [x>0]), true \wedge [x>0])$

$= SP([x>0]), SP(y=2;, [x>0])$

$= (\exists y' : x>0 \wedge y=2) \wedge [x>0]$

$= [x>0] \wedge [y=2]$ SAT e.g. $x=1$, $y=2$ #

$G_3 = SP([:(x>0)], y=-1;, [y==0]), true)$

$= SP(y=-1;, [y==0]), SP([x \leq 0], true)$

$= SP(y=-1;, [y==0]), true \wedge [x \leq 0]$

$= SP([y==0]), SP(y=-1;, [x \leq 0])$

$= \exists y' : y==0 \wedge y=-1$

↳ contradicted ↳ not SAT #

Exercise 2

$SP((x=x+1;, x>0, x=x+y;), x_0=0)$

$\Rightarrow x_0=0 \wedge x_1=x_0+1 \wedge x_1>0 \wedge x_2=x_1+y_0$

$G_1 := ([x>0], y=1;, [x<=0])$

$\Rightarrow SP(G_1, true)$ contradicts!

$\Rightarrow true \wedge [x_0>0] \wedge y_0=1 \wedge [x_1<=0]$ not SAT #

$G_2 := ([x>0], y=2;, [x>0])$

$\Rightarrow SP(G_2, true)$

$\Rightarrow [x_0>0] \wedge [x_1>0] \wedge y_0=2$

SAT, Satisfiable $\Rightarrow$ e.g. $x=1, y=2$ #

$G_3 := ([!(x>0)], y=-1;, [y==0])$

$\Rightarrow SP(G_3, true)$

$\Rightarrow [y_0==0] \wedge x_0<=0 \wedge y_1=-1$

SAT e.g. $x=-1, y=-1$ #

$G_4 := (y=0;, [x>0], y=5;, [x>10])$

$\Rightarrow SP(G_4, true)$

$\Rightarrow [x_0>10] \wedge y_0=0 \wedge x_1>0 \wedge y_1=5$

SAT $x=11$ $y_1=5$

#

練習1 Strongest Post

假定: ① $SP([expr], f) = f \wedge expr$ 這裡有個 v' 存在

② $SP(v := expr;, f) = \exists v' : f[v \to v'] \wedge v = expr[v \to v']$

在這裡比較複雜的是，我們必須把分号化前的分号之狀態給改名，這樣才可以區別 v 表示這個分号之最末了的狀況。
→ 也就是說，中文分号之會被改成 v'。

所以想当然心，我們可以把上述的法簡化的治伸成一個操作序列，所以就可以創造下列路徑公式（程式路徑的）

- ③ $SP(\varepsilon, f) = f$ → 空序[]
- ④ $SP((op_1, \cdots, op_n), f) = SP((op_2, \cdots, op_n), SP(op_1, f))$

現在就來一分練習，假設操作序列 G_i.

計算 Strongest Post Condition $SP(G_i, true)$，並判斷下列困式是否符合。

a) G_1: ([x>0], y=1;, [x<=0])

$SP(G_1, true) = SP([x>0], y=1;, [x<=0], true)$ 公式①

$= SP(y=1;, [x<=0], SP([x>0], true))$

$= SP(y=1;, [x<=0], x>0)$

$= SP([x<=0], SP(y=1;, x>0))$ 公式① 改名x'!

$= SP([x<=0], x>0 \wedge y=1)$ 公式③

$= x>0 \wedge y=1 \wedge x \le 0$ (UNSAT) 不符合

b) G_2: ([x>0], y=2;, [x>0])

$SP(G_2, true) = SP([x>0], y=2;, [x>0], true)$ 公式①

$= SP(y=2;, [x>0], SP([x>0], true))$

$= SP(y=2;, [x>0], x>0))$

$= SP([x>0], SP(y=2;, x>0))$

$= SP([x>0], x>0 \wedge y=2))$ 公式③

$= x>0 \wedge y=2 \wedge x>0$ (SAT) 符合

c) G_3: ([!(x>0)], y=-1;, [y==0])

$SP(G_3, true) = SP([!(x>0)], y=-1;, [y==0], true)$ 公式①

$= SP(y=-1;, [y==0], SP([!(x>0)], true))$

$= SP(y=-1;, [y==0], \neg(x>0))$ 公式②

$= SP([y==0], \neg(x>0) \wedge y=-1)$ 公式③ 矛盾

$= \neg(x>0) \wedge y=-1 \wedge y=0$ (UNSAT) 不符合

d) G_4: (y=0;, [x>0], y=5;, [x>10])

$SP(G_4, true) = SP(y=0;, [x>0], y=5;, [x>10], true)$ 公式①

$= SP([x>0], y=5;, [x>10], SP(y=0;, true))$

$= SP([x>0], y=5;, [x>10], y=0)$

$= SP(y=5;, [x>10], SP([x>0], y=0))$ 公式①

$= SP(y=5;, [x>10], y=0 \wedge x>0)$

$= SP([x>10], SP(y=5;, y=0 \wedge x>0))$ 公式② → y=0 跟 y=5; 出矛盾，所以要把改的那一個改名為 y'

$= SP([x>10], \exists y' : y'=0 \wedge x>0 \wedge y=5)$ 有個 y'=0 存在 ⇒ 恆 true，因為我們只討論 y，因此 y'=y' 恆成立。

$= SP([x>10], x>0 \wedge y=5)$

練習2 Skolemization via Single Static Assignment (SSA)

利用狀態賦值的資料流變化
(同步，但也可以在時間比較短的時候用非同步)

這個公式從上一題練習所提到的「有可能包含太多的存在量化的變數」，因由這些量化變數有時候會很難分辨出執行順序以及事件發生的順序。
有一個方法叫SSA，可以表示這些情況。利用在每次都給新的ID。這樣可以強制每個ID只能被賦值一次。這種編譯方式在編譯器上也是非常常見，這樣也可以更快速地驗證程式。
一個新的ID通常都會用下標形的數字來命名，因此可以變得更清楚。

例：SP(([①] $x=x+1$; [③] $x>0$, [④] $x=x+y$;), [①] $x_0=0$)
可以改寫成(利用前述的parsing順序)成SSA模式 → 因為true在邏輯及"true"的特性 所以不會影響結果，且直接省略(下列直接省略)
$x_0=0 \wedge x_1=x_0+1 \wedge x_1>0 \wedge x_2=x_1+y_0$ → 賦值後加1之故

請用最強事前條件(Strongest postcondition) SP(g_i, true)，並用SSA形來判斷下列函式何者滿足。

a) G_1:([①] [x>0], [②] y=1;, [③] [x<=0]) $x_0>0 \wedge y_1=1 \wedge x_0 \wedge x_0 \leq 0$ (UNSAT) 不符合 — 矛盾

b) G_2:([①] [x>0], [②] y=2;, [③] [x>0]) $x_0>0 \wedge y_1=2 \wedge x_0>0$ (SAT) 不符合 — 重複

c) G_3:([①] [!(x>0)], [②] y=-1;, [③] [y==0]) $\neg(x_0>0) \wedge y_1=-1 \wedge y_1=0$ (UNSAT) 不符合 → 條件不一樣但一樣的變數，且矛盾

d) G_4:([①] y=0;, [②] x>0, [③] y=5;, [④] x>10) $y_1=0 \wedge x_0>0 \wedge y_2=5 \wedge x_0>10$ (SAT) 符合

重點整理：
滿足性檢查
① 判斷順序
② 賦值時給予代號
③ 確認矛盾是否存在

練習3 有界模型確認法 (Bounded Model Checking) (非同步 asynchronous)

最強事前條件可以建立一個非常好的模型確認分析。
SP常常是有界模型確認法中的一個環節，所有的有界的程式路徑都會被確認。
在這種情況下，我們可以假定一個建立程式分析 $\mathbb{F}$，包含著：

x	y	IFF	IMP
0	0	1	1
0	1	1	0
1	0	0	0
1	1	1	1

1. 抽象空間 $D_{\mathbb{F}}=(C,\mathcal{E},[\![\cdot]\!])$ 包含
   ① 固定狀態集合 C
   ② 半束 $\mathcal{E}=(F, \sqsubseteq_F, \sqcup_F, T)$
   - F → 程式分析函式集合
   - $\sqsubseteq_F$ → 歸類關係：$\phi \sqsubseteq_{\mathbb{F}} \psi \Leftrightarrow (\phi \Rightarrow \psi)$ (⇔：IFF 相當若；⇒：Implies)
   - T → 恒真式 例：恒正函數
   ③ 固化函數 $[\![\cdot]\!]$ → $[\![\phi]\!]=\{c \mid c\in C \wedge c\models\phi\}$

注意：這可能會有語法上不同 (syntactically)，語意上 (semantically) 其實是相同的函式限定。
我們會多多少少用這些函數來表示子集C，換句話說：如果 $[\![\phi]\!]=[\![\psi]\!]$ 則等同於 $\phi\Leftrightarrow\psi$ 的意思。

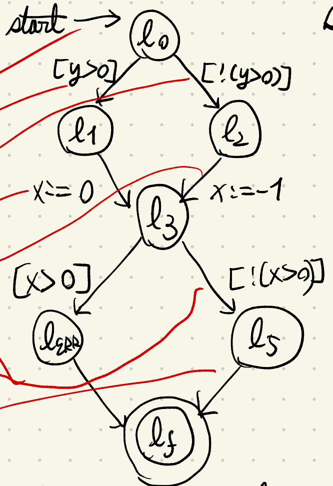
2. 轉換關係 $\rightsquigarrow_{\mathbb{F}}$ 如果 $g=(\ell, op, \ell')$ 就會包含 $\phi \rightsquigarrow \phi'$ 且保持 $SP(op,\phi)=\phi'$
   等同於表示這個函式語法是由最強事前條件法所創造
   (避免處理那些語法性相同的後綴函數們)
   我們可以用SAT檢查來進一步的確認 ϕ' 恆成立
   但做時，SAT常需要大量資源(例：只在錯誤/目標狀態執行
   但為了保持精簡，只討論那些
   只有符合 $\phi \rightsquigarrow \phi'$ 的 ϕ')

3. 合併的算子定義為 $merge_{\mathbb{F}}(\phi,\psi)=\psi$ (例：禁合併新狀態的時候)
4. 停止的算子定義為 $stop_{\mathbb{F}}=false$ (例：禁停止的時候)

由上述練習1跟2提及的，現在來進行試做。

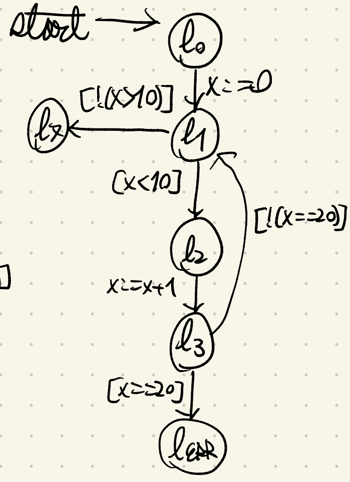
a) 執行CPA演算法
建立程式分析 $\mathbb{L}\times\mathbb{F}$ 在図1
利用 $e_0=(l_0,\top)$ 為初始狀態
SSA形跟最強事前條件法都可以使用
形式化法：

$$
\begin{aligned}
reached=\{&(l_0,\top),\\
&(l_1, y_0>0),\\
&(l_2, \neg(y_0>0)),\\
&(l_3, (y_0>0)\land x_1=0),\\
&(l_3, \neg(y_0>0)\land x_1=-1),\\
&(l_5, \neg(y_0>0)\land x_1=-1\land\neg(x_1>0)),\\
&(l_5, (y_0>0)\land x_1=0\land\neg(x_1>0)),\\
&(l_f, \neg(y_0>0)\land x_1=-1\land\neg(x_1>0))\\
\}&
\end{aligned}
$$



CFA $A=(L,\mathcal{E},l_0)$

図1



CFA $A=(L,\mathcal{E},l_0)$

図2

依照図1沒有loop的狀況都沒問題，但如果像図2的話，就沒辦法保證程式一定會終止了。
比較好的理由就是，想要在一定的最長路徑長度到達探索極限。
然而，我們可以利用CPA $\mathbb{B}$，就會分成以下4部分：

1. 抽象空間 $D_{\mathbb{B}}=(C,\mathcal{E},[\![\cdot]\!])$ 由
   - 集合<固定狀態>： $C=[\![a]\!]$
   - 固化函數： $[\![\cdot]\!]$
   - 半束(格)： $\mathcal{E}=(\mathbb{N}\cup\{\top_{\mathbb{B}}\},\sqsubseteq_{\mathbb{B}},\sqcup_{\mathbb{B}},\top_{\mathbb{B}})$
     - $\{\top_{\mathbb{B}}\}$ 用 → 符合試的比
     - $\mathbb{N}$ → 自然數 (aka. 非負整數)

2. 轉換關係 $\rightsquigarrow_{\mathbb{B}}$ 包含 $a\overset{g}{\rightsquigarrow}b'$
   如果 $\begin{cases} b=a-1\\ b\geq 0\end{cases}$ → 為了停止探索之用
   (因為如果到0就不會有反轉換關係的安全性被破壞)
3. 合併的算子定義為 $merge_{\mathbb{B}}(a,b)=b$ (例：不合併的時候)
4. 停止的算子定義為 $stop_{\mathbb{F}}=stop^{sep}$

b) 執行CPA演算法
   - 可建立程式分析： $\mathbb{L}\times\mathbb{F}\times\mathbb{B}$ 図2
   - 初始狀態： $e_0=(l_0,\top,10)$
   - SSA形跟最強事前條件法都可以使用

$$
\begin{aligned}
reached=\{&(l_0,\top,10), && x:=0\\
&(l_1, x_1=0, 9), && [x<10]\\
&(l_2, x_1=0\land x_1<10, 8), && x:=x+1\\
&(l_3, x_1=0\land x_1<10\land x_2=x_1+1, 7), && [!(x==20)]\\
&(l_1, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20, 6), && [x<10]\\
&(l_2, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20\land x_2<10, 5), && x:=x+1\\
&(l_3, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20\land x_2<10\land x_3=x_2+1, 4), && [!(x==20)]\\
&(l_1, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20\land x_2<10\land x_3=x_2+1\land x_3\neq 20, 3), && [x<10]\\
&(l_2, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20\land x_2<10\land x_3=x_2+1\land x_3\neq 20\land x_3<10, 2), && x:=x+1\\
&(l_3, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20\land x_2<10\land x_3=x_2+1\land x_3\neq 20\land x_3<10\land x_4=x_3+1, 1), && [!(x==20)]\\
&(l_1, x_1=0\land x_1<10\land x_2=x_1+1\land x_2\neq 20\land x_2<10\land x_3=x_2+1\land x_3\neq 20\land x_3<10\land x_4=x_3+1\land x_4\neq 20, 0)
\end{aligned}
$$

iter₁ 第一代₁ ; iter₂ 第二代₂ ; iter₃ 第三代₃