Exercise 02

2021/10/22

Supplemental Material

Please read Sections 16.3.1–16.4.3 of Combining Model Checking and Data-Flow Analysis.

This sheet provides exercises to the following topics:

a) Data-Flow Analysis (Sect. 16.3.2)

b) Model Checking (Sect. 16.3.3)

c) Configurable-Program-Analysis Algorithm (CPA Algorithm) (Sect. 16.3.4–16.3.5)

d) Constant-Propagation Analysis (Sect. 16.4.2)

e) Reaching-Definitions Analysis (Sect. 16.4.3)

As background, we additionally need the concept of a control-flow automaton from Sect. 16.3.1.

Our primary goal

In a large part of the lecture, we will be concerned with *reachability analysis*. Our goal is to prove or disprove that a program P under analysis fulfills a given safety property φ, formally, $P \vdash \varphi$.

Safety properties encode that something may never happen. For example, "The program never calls a failing assert". To prove $P \vdash \varphi$ for such a safety property, we have to compute all reachable program states of P and show that none violate φ. This is called *reachability analysis*.

Since there is, potentially, an infinite amount of reachable concrete program states, we use abstraction to compute them efficiently. The used abstraction is defined by an *abstract domain* using a lattice of abstract states. Each of the abstract states represents a set of concrete program states. If all computed, reachable abstract program states only represent concrete program states that fulfill φ, we know that $P \vdash \varphi$.

For proving a property, it is necessary to consider all reachable program states. If a computed abstraction fulfills a property but describes (due to its abstraction) a superset of the concrete program states that are actually reachable, this is ok. This is called *over-approximation*. Almost all of the techniques we consider are over-approximations.

For disproving a property, it is enough to find a single counterexample and not all reachable program states must be computed. But the computed counterexample must be actually reachable, otherwise it is invalid. This is called *under-approximation*. One example for under-approximation is software testing.

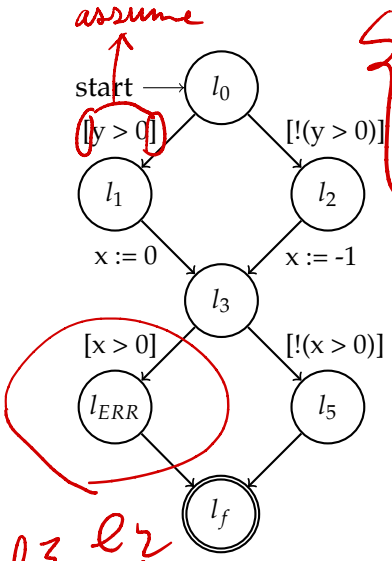
More Info

Both the CPAs for constant propagation and reaching definitions from the book chapter are abstract analyses that can be used stand-alone, because their abstract domains encode the current program location. But in general, the CPA concept allows to compose multiple CPAs with a Composition CPA (Sect. 16.3.6). This enables separation of concerns and information exchange between different abstract domains. With the use of a Composition CPA, the dedicated *location analysis* (Sect. 16.4.1) can be used to track the current location in the program, and other analyses can focus on their specific domain.

1 Data-flow Analysis and Model Checking with Constant Propagation

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

Above you see the program under analysis and the corresponding CFA A.
We write `[p]` as synonym for `assume(p)`.

Let our property be φ = "Label ERR is never reached".

1. Use data-flow analysis to check $A \vdash \varphi$:
   a) Apply the data-flow analysis algorithm $DFA(L, E, \rightsquigarrow, e_0)$ with the constant-propagation analysis CO to CFA A. This means:
      - $\rightsquigarrow = \rightsquigarrow_{CO}$
      - $e_0 = (l_0, \top)$

      List the set `reached` of returned abstract states.
   b) Do all abstract states in `reached` fulfill φ? State whether property φ is fulfilled or violated according to the applied algorithm.
2. Use model checking to check $A \vdash \varphi$:
   a) Apply the model-checking algorithm $Reach(L, E, \rightsquigarrow, e_0)$ with the constant-propagation analysis CO to CFA A. This means:
      - $\rightsquigarrow = \rightsquigarrow_{CO}$

- $e_0 = (l_0, v_\top)$

List the set `reached` of returned abstract states.

b) Do all abstract states in `reached` fulfill φ? State whether property φ is fulfilled or violated according to the applied algorithm.

2 CPA Formalism

1. List the four components of a CPA.
2. List four possible waitlist iteration orders that may make sense in the CPA algorithm. For each iteration order you list, state one potential advantage and one potential disadvantage.
3. CPAs can be built to both over-approximate and under-approximate the program-state space. In Section 2.1, the following publication contains all properties that must be fulfilled by a CPA to be a sound over-approximation:
   Dirk Beyer, Thomas A. Henzinger, and Grégory Théoduloz. Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis, CAV 2007.

   a) List the soundness criteria that must be fulfilled by the abstract domain and the transfer relation.

   b) List the soundness criteria that must be fulfilled by the *merge* operator.

   c) List the soundness criteria that must be fulfilled by the *stop* operator.
4. Define a *merge* operator for the constant-propagation analysis that will lead to an under-approximation of the program-state space.

3 Constant Propagation as CPA

1. Compute the result of the following statement:

   a) $(l, \{x \mapsto 2, y \mapsto 0\}) \sqcup (l, \{x \mapsto 2, y \mapsto 3\})$

   b) $(l, \{x \mapsto 2, y \mapsto 0\}) \sqcup (l, \{x \mapsto \top_{\mathbb{Z}}, y \mapsto 2\})$

   c) $(l, \{x \mapsto 2, y \mapsto 0\}) \sqcup (l', \{x \mapsto 2, y \mapsto 0\})$ for $l \neq l'$
2. Use the CPA algorithm to check $A \vdash \varphi$ (for the CFA and property of the first exercise):

   a) Apply the CPA algorithm $CPA(\mathbb{D}, e_0)$ with the constant-propagation analysis $\mathbb{CO}$ to CFA A. This means:

   - $\mathbb{D} = \mathbb{CO}$
   - $e_0 = (l_0, v_\top)$

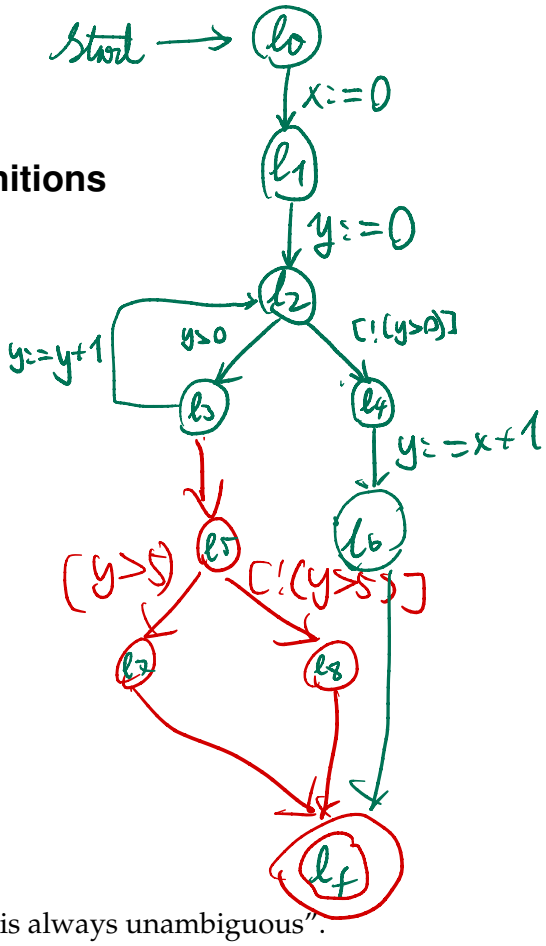
   List the set `reached` of returned abstract states.

   b) Do all abstract states in `reached` fulfill φ? State whether property φ is fulfilled or violated according to the applied algorithm.

4 CPA Algorithm with Reaching Definitions

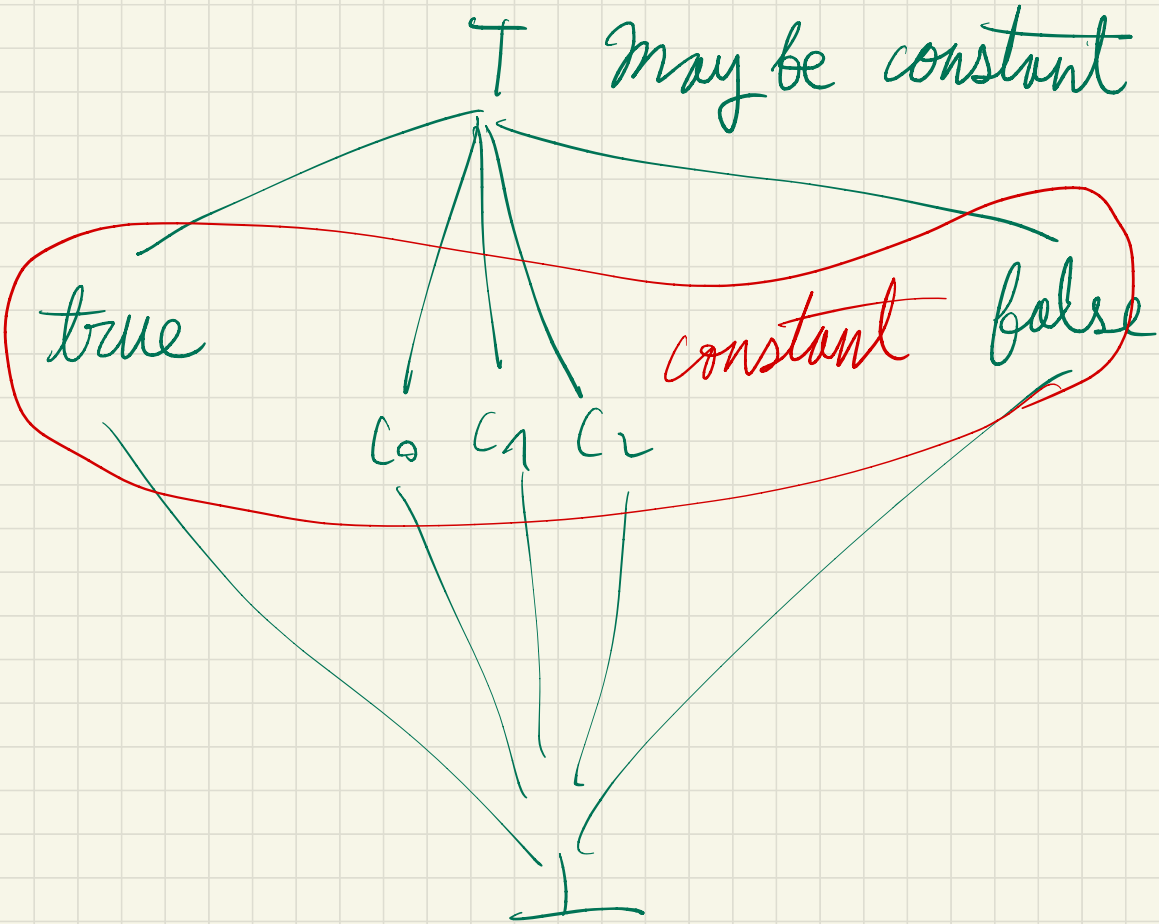
```
1  int x = 0;
2  int y = 0;
3  while (y > 0) {
4    y = y + 1;
5  }
6
7  if (y > 5) {
8    x = 0;
9  } else {
10   x = -1;
11 }
12
13 y = x + 1;
```

Program 2



Above you see the program under analysis.
Let our property be φ_2 = "The last definition of x is always unambiguous".

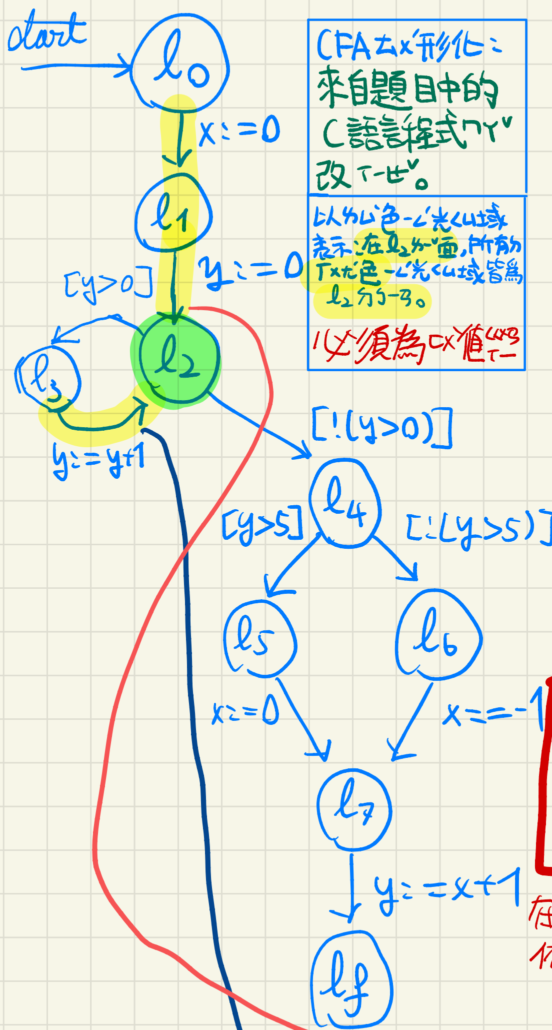
1. Draw the CFA A_2 for Program 2.
2. Use the CPA algorithm to check $A_2 \vdash \varphi_2$:
   a) Apply the CPA algorithm $CPA(\mathbb{RD}, e_0)$ with the reaching-definitions analysis $\mathbb{RD}$ to CFA A_2. Assume $e_0 = (l_0, \emptyset)$, where l_0 is the initial state of A_2.
      i. List the set `reached` of returned abstract states.
      ii. How often did the *stop* operator return true?
   b) Do all abstract states in `reached` fulfill φ_2? State whether property φ_2 is fulfilled or violated according to the applied algorithm.
3. Which composition of CPAs (that you know) could be used to correctly solve $A_2 \vdash \varphi_2$?



$$\text{any} \sqcap \top = \text{any}$$

$$\sqcap \bot = \bot$$

$$c_i \sqcap c_j = \begin{cases} c_i & \text{if } i = j \\ \bot & \text{if } i \neq j \end{cases}$$



程式入口 l → location

一ㄍ CFA A = (L, l0, G) 表示一個簡程

L → 位置集合；G → ㄅㄧㄢ集合 表示集法

G ⊆ L × Ops × L

Ops → 運算子集合

本題：{:=, >}（:= ㄈㄨˋ值）

一ㄍㄈㄨˋ值ㄈㄢˋㄉㄨㄟˋ ㄅㄧㄢ，固有的程式狀ㄊㄞˋ。

例：一ㄍㄅㄧㄢ ㄅㄧㄠ ㄕˋ作 [y>5] 成立。當這ㄍ程式狀態不符合這ㄍ條件時，程式就ㄈㄢˋ ㄅㄧㄢ 一 ㄍ開。

抵ㄉㄚˊ集合 reached = {(l0, ∅), …… 起ㄉ一ㄢ前ㄨˊ ㄅㄧㄢ（ㄈㄨˋ值）

（不從 l0 到 l1 的所有 assignment）

(l1, {(x, l0, l1)}), → G：所謂ㄉ一ㄝ (ㄅㄧㄢ)

(l2, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}),

(l3, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}),

(l4, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}),

(l5, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}),

(l6, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}),

(l7, {(x, l5, l7), (y, l6, l7), (y, l1, l2), (y, l3, l2)}),

(lf, {(x, l5, l7), (x, l6, l7), (y, l7, lf)}}

在抵ㄉㄚˊ集合中，最後四ㄍ狀ㄊㄞˋ就ㄈㄚ生了合併。

依照合併公式：merge(e', e'')

e'：新算好ㄉ後ㄐ一ㄝˋ；e''：ㄐ一ㄡˋ有抵ㄉㄚˊ集合ㄉ狀ㄊㄞˋ。e'' ∈ reached

「合併」在 e' 跟 e'' ㄈㄚ生ㄅㄧㄢ化時ㄘㄞˊㄈㄚ。

merge_RD(e', e'')

= merge_RD((l2, {(x, l0, l1), (y, l3, l2)}), (l2, {(x, l0, l1), (y, l1, l2)})) ∪ ㄅㄧㄢ集

= (l2, {(x, l0, l1), (y, l1, l2), (y, l3, l2)})

第二次合併會稍微複ㄗㄚˊ一ㄉ一ㄢ：

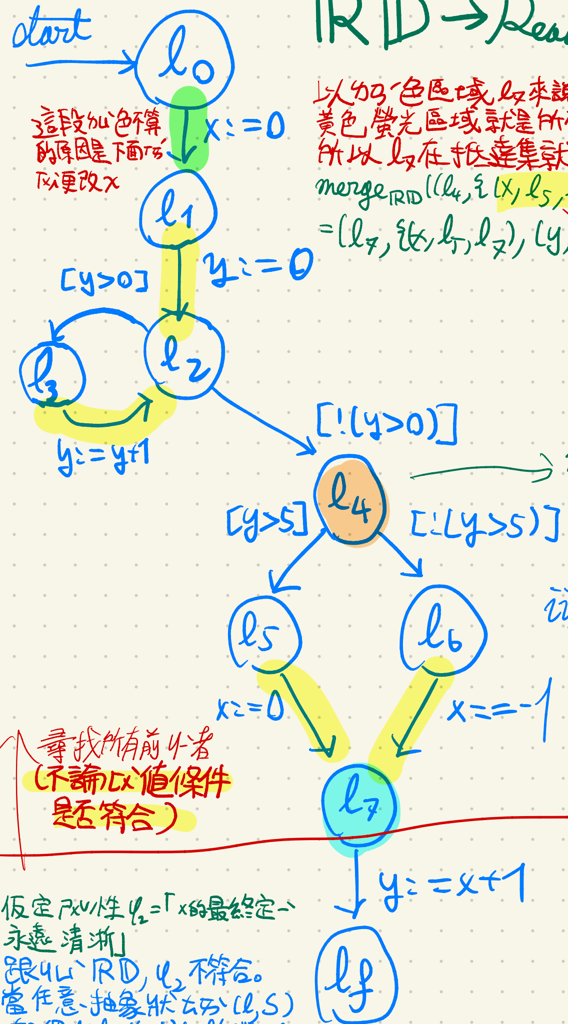
後ㄐ一ㄝˋ者：e' = (l3, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}) 已經跟既有狀ㄊㄞˋ合併了 e'' = (l3, {(x, l0, l1), (y, l1, l2)})

這就造成 $e_{new} = e'$ 且 $e_{new} \neq e''$ 所以我們要在終結檢查 stop 前，把 e_{new} 加進 CPA 一ㄉ算法的抵達集和等待集裡面。

merge_RD((l3, {(x, l0, l1), (y, l1, l2), (y, l3, l2)}), (l3, {(x, l0, l1), (y, l1, l2)}))

= ((l3, {(x, l0, l1), (y, l1, l2), (y, l3, l2)})

RD → Reaching Definitions 抵ㄉㄚˊ定ㄧˋ們



以ㄌㄩˋ色區域 l_7 來說：
黃色螢光區域就是所有可行ㄅㄨˋ-ㄙ(皆為ㄘㄨˋ值ㄕˋ件)
所以 l_7 在抵達集就是：

$merge_{RD}((l_4,\{(x,l_5,l_7),(y,l_3,l_2),(y,l_1,l_2)\}),(l_4,\{(x,l_6,l_7),(y,l_3,l_2),(y,l_1,l_2)\}))$
$=(l_7,\{(x,l_5,l_7),(y,l_1,l_2),(y,l_3,l_2),(x,l_6,l_7)\})$

→依照ㄅ-ㄙ的特性：一次只能行駛一個路徑。

尋找所有前ㄐㄧˋ者
(不論ㄘㄨˋ值條件是否符合)

G(l_3,l_2) 發生&不發生的合併

$merge_{RD}((l_4,\{(x,l_0,l_1),(y,l_3,l_2),(y,l_1,l_2)\}),(l_4,\{(x,l_0,l_1),(y,l_1,l_2)\}))$
$=(l_4,\{(x,l_0,l_1),(y,l_3,l_2),(y,l_1,l_2)\})$

ii) stop(ㄓㄢˋ停函ㄕˋ)ㄏㄨㄟˋ回ㄔㄨㄢˊㄐㄧˇ次 true？

→第6行(演算法偽碼)

至少四次——因為ㄈㄚ生四次合併
在CPA演算法，計算出了新的後ㄐㄧˋ者：e'，
然後 e' 就會去跟抵ㄉㄚˊ集所有其他的位置合併，
最後執行 stop(停止函ㄕˋ)來
檢查後ㄒㄩˋ終止(13行)

e'' ∈ reached
(ㄐㄧㄢˋ 12行)

假定ㄕˋ性 φ_2 =「x的最終定ㄧˋ永遠清晰」
跟ㄐㄧㄡˋ RD，φ_2 不符合。
當任意抽象狀ㄊㄞˋ(l,S)
有2個 tuples(ㄒㄩ ㄌㄧㄝˋ) $(x,l',l'')\in S$。
在這題的話，就是 (l_7,S) 跟 (l_f,S)。

請注意，這個只是假象：→程式OK！但分析回報錯誤
φ_2 其ㄕˊ沒有不符合。
在程式2(上圖)，第7行只會在 $y\leq 0$ 抵ㄉㄚˊ得了。

RD 沒有任何法去ㄐㄧˋ住ㄅㄧㄢˋㄕㄨˋ的值[?]，
所以也沒辦法去ㄐㄧㄢˇ少集資訊[?]。

CPA的組成，有哪些可以正確解出 $A_2\vdash\varphi_2$
可以利用RD定ㄧˋ們，利用CO(常ㄕˋ傳ㄉㄧˋ)，
來進行合併。

【ㄈㄢˋ例：】
設一ㄍㄜ CFA ㄅ-ㄙ $(l_3, y:=y+1, l_2)$
即其CPA演算法：

A. 計算 $e'=(l_2,\{(x,l_0,l_1),(y,l_3,l_2)\})$
B. 執行合併 $merge(e',e'')$，且 $e''=(l_2,\{(x,l_0,l_1),(y,l_3,l_2)\})$
屆時創造新狀ㄊㄞˋ(合併後的) $e_{new}=(l_2,\{(x,l_0,l_1),(y,l_1,l_2),(y,l_3,l_2)\})$
C. 從等待清單抵ㄉㄚˊ集中把 e'' 改成 e_{new}
D. 如果 e' 已經在 reached 裡面了，則檢查 stop(e', reached) 的回ㄔㄨㄢˊ值。小於等於(來ㄅㄧˇ較用)
因為 $e'\sqsubseteq e_{new}$
且 $e_{new}\in$ reached
則停止ㄙㄨㄢˋ子在每次合併時，回ㄔㄨㄢˊ true。

CPA結構：轉ㄏㄨㄢˇㄍㄨㄢ係

CPA(D, ⤳, merge, stop)

- D → 抽象領域
- ⤳ → 轉ㄏㄨㄢˇㄍㄨㄢ係
- merge → 合併運算子通常ㄉㄧㄢˊㄐㄧˊ
- stop → 終結檢查

有4ㄓㄨㄥˇ等待集ㄉㄧㄝˊ代順序在CPA演算法：

a) DFS (深度ㄧㄡ先搜尋) depth first search

ㄧㄡ ㄉㄧㄢˇ：

i) 抵ㄉㄚˊ「終ㄉㄧㄢˇ」快速。

ii) ㄉㄨˊ空路徑抵ㄉㄚˊ終ㄉㄧㄢˇ。

iii) 在可抵ㄉㄚˊㄉ程式狀ㄊㄞˋ中，有更高的ㄐㄧ率可以找到程式的深ㄘㄥˊ問ㄊㄧˊ。

ㄑㄩㄝ ㄉㄧㄢˇ：

i) ㄨˊ法快速ㄉㄚˊ成相路徑的「ㄍㄨㄤˇ泛ㄓ佈」

ii) 分析時可能ㄏㄨㄟˋ卡住。

b) BFS (ㄍㄨㄤˇ度ㄧㄡ先搜尋) Breadth first search

ㄧㄡ ㄉㄧㄢˇ：

i) 快速ㄉ抵ㄉㄚˊ廣度ㄘㄥˊ級。

ii) 同時抵ㄉㄚˊ所有路徑。

iii) 在程式起ㄉㄧㄢˇ附近，快速找到問題。

ㄑㄩㄝ ㄉㄧㄢˇ：

i) 需要花一些時間ㄔㄨˋ理到終ㄉㄧㄢˇ的「路徑探索」

c) Reverse-post order (反後序法《模型檢索》§16.3.5)

ㄧㄡ ㄉㄧㄢˇ：　*meet → lower bound

i) 減少所需的重複計算次ㄕㄨˋ，因為如果所有前ㄐㄧˋ者們計算完成後，CPA只ㄏㄨㄟˋ從lower bound ㄐㄧˋㄒㄩˋ。

→ 在ㄐㄧˋㄒㄩˋ前，狀ㄊㄞˋ合併即全ㄅㄨˋ完成。

ㄑㄩㄝ ㄉㄧㄢˇ：

i) 需要花一些時間ㄔㄨˋ理到終ㄉㄧㄢˇ的「路徑探索」

d) Random (ㄙㄨㄟˊㄐㄧ法)　*skewed → ㄆㄧㄢ重ㄉ

ㄧㄡ ㄉㄧㄢˇ：不偏重於深度探索或ㄍㄨㄤˇ度探索。

ㄑㄩㄝ ㄉㄧㄢˇ：成功率也是ㄙㄨㄟˊㄐㄧㄉ。

e) Random weighted (ㄙㄨㄟˊㄐㄧ加ㄑㄩㄢˊ)

ㄙㄨㄟˊㄐㄧ ㄒㄩㄢˇㄗㄜˊ 一ㄍㄜˋ ㄌㄧˊ入口很遠ㄉ跟ㄌㄧˊ入口很近ㄉ抽象狀ㄊㄞˋ，但ㄧㄡ先位置都ㄏㄨㄟˋ在程式入口附近。　*favor → ㄆㄧㄢㄏㄠˋ

ㄧㄡ ㄉㄧㄢˇ：ㄆㄧㄢ向ㄑㄧㄢˇ層探索，但並不ㄏㄨㄟˋ完全ㄨˊ視深ㄘㄥˊ探索。

ㄑㄩㄝ ㄉㄧㄢˇ：深ㄘㄥˊ探索ㄉ最佳ㄑㄩㄢˊ重
vs
底ㄉㄨˋ完成分析ㄉㄨˋ全程式ㄉㄑㄧㄢˇ化探索

CPA的soundness criterias (健全性ㄅㄧㄠ準)

(ㄅㄧˋㄒㄩ符合抽象領域跟ㄓㄨㄢˇㄏㄨㄢˋㄍㄨㄢㄒㄧ)

i) $[\![\top]\!] = C$ 跟 (都存在) $[\![\bot]\!] = \emptyset$

ii) $\forall e, e' \in \mathcal{E}: [\![e \sqcup e']\!] \supseteq [\![e]\!] \cup [\![e']\!]$

iii) $\forall e \in \mathcal{E}: \exists e' \in \mathcal{E}: e \rightsquigarrow e'$

iv) $\forall e \in \mathcal{E}, g \in \mathcal{G}: \bigcup_{e \overset{g}{\rightsquigarrow} e'} [\![e']\!] \supseteq \bigcup_{c \in [\![e]\!]} \{c' \mid c \xrightarrow{g} c'\}$

(必須在合併運算子符合)

i) $\forall e, e' \in E: e' \sqsubseteq merge(e, e')$

(必須在停止運算子符合)

i) $stop(e, R) = true \Rightarrow [\![e]\!] \subseteq \bigcup_{e' \in R} [\![e']\!]$

定ㄧˋ CPA ㄧㄍ在程式狀ㄊㄞˋ空間ㄉ合併運算子

$$merge((l, v), (l', v')) = \begin{cases} (l', v) & \text{if } l = l' \\ (l', v') & \text{otherwise} \end{cases}$$

(v, v'：抽象ㄉㄧㄢˇㄕㄨˋ值)

合併動作：

1. 從等待清單中移除 (l', v')
2. 如果沒有任何 (l', v) 的後ㄐㄧˋ者被計算，則 (l', v) 就會在可抵ㄉㄚˊ狀ㄊㄞˋ中的估算空間內。

CPA → Constant-propagation analysis → 常ㄕㄨˋㄔㄨㄢˊ紀 分析

CPA的常ㄕㄨˋㄔㄨㄢˊㄉㄧˋ

ㄈㄨˋ值狀ㄊㄞˋ → 上界算ㄈㄨˊ (upper bound operator)

1. a) $(\ell, \{x \mapsto 2, y \mapsto 0\}) \sqcup (\ell, \{x \mapsto 2, y \mapsto 3\}) = (\ell, \{x \mapsto 2, y \mapsto \top_Z\})$

b) $(\ell, \{x \mapsto 2, y \mapsto 0\}) \sqcup (\ell, \{x \mapsto \top_Z, y \mapsto 2\}) = (\ell, \{x \mapsto \top_Z, y \mapsto \top_Z\}) = (\ell, v_\top)$

c) $(\ell, \{x \mapsto 2, y \mapsto 0\}) \sqcup (\ell', \{x \mapsto 2, y \mapsto 0\})$ for $\ell \neq \ell' = (\top_L, \{x \mapsto 2, y \mapsto 0\})$

任意

2. 套用CPA演算法來檢查 $A \vdash \varphi$ (為3CFA跟第一ㄍㄜ練習的ㄕㄨˇ性們):

a) 套用CPA演算法 $CPA(\mathbb{D}, e_0)$ 跟CFA A的常ㄕㄨˋㄔㄨㄢˊㄉㄧˋ $\mathbb{CO}$，也就是說:

- $\mathbb{D} = \mathbb{CO}$
- $e_0 = (\ell_0, v_\top)$

請列出回傳的抽象狀ㄊㄞˋ集:

reached $= \{(\ell_0, v_\top),$
$(\ell_1, v_\top),$
$(\ell_2, v_\top),$
$(\ell_3, v_\top),$
$(\ell_{ERR}, v_\top),$
$(\ell_5, v_\top),$
$(\ell_f, v_\top)\}$

* φ = 「錯誤標ㄑㄧㄢ永不抵ㄉㄚˊ」

reached裡面的所有抽象狀ㄊㄞˋ都符合 φ 嗎?

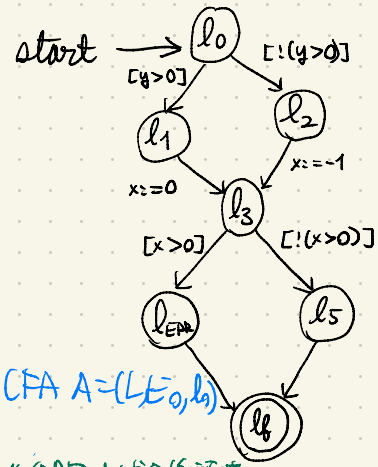
請列出是否ㄕㄨˇ性 φ 滿足或違反，根ㄐㄩˋ CPA演算法。

不行。抽象狀ㄊㄞˋ $(\ell_{ERR}, v_\top)$ 違反 φ，因為這個抽象狀ㄊㄞˋ在ERR標ㄑㄧㄢ上。

根ㄐㄩˋ上述原因，φ 違反3DFA演算法

(注意:這是個錯誤ㄐㄧㄥˇ報)

程式OK!但分析回報說錯誤位置是可抵ㄉㄚˊ的。



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

我們用上述分析語法
[P] 來表示 assume(p)

假設我們要 φ = "Label ERR is never reached" 檢查 $A \vdash \varphi$: 要不一定或成立

a) 用(資料流分析演算法) data-flow analysis DFA $(L,E,\leadsto,e_0)$, 且利用前段所學習分析 CFA 到 CPA。

這就代表:
- $\leadsto = \leadsto_{CPA}$
- $e_0 = (l_0, v_\top)$

請列出回傳的抽象狀態集 reached:

$$\text{reached} = \{(l_0,(l_0,v_\top)),\ (l_1,(l_1,v_\top)),\ (l_2,(l_2,v_\top)),\ (l_3,(l_3,v_\top)),\ (l_{ERR},(l_{ERR},v_\top)),\ (l_5,(l_5,v_\top)),\ (l_f,(l_f,v_\top))\}$$

在最終抵分集, 所有的抽象狀態都被 $v_\top$ 值 (因為沒有確切的已知 v 值)

l_3 有 join (上界 UB) 的原因 (DFA 310)

$$\text{reached}(l_3) = (l_3,\{x\mapsto 0, y\mapsto \top_{\mathbb{Z}}\}) \sqcup (l_3,\{x\mapsto -1, y\mapsto \top_{\mathbb{Z}}\}) \quad \text{未定義}$$
$$= (l_3,\{x\mapsto \top_{\mathbb{Z}}, y\mapsto \top_{\mathbb{Z}}\}) \quad x,y \text{都在 top}$$
$$= (l_3, v_\top)$$

b) 所有的抽象狀態都符合 φ 嗎?
請說明是否成立 φ 有沒有符合或違反演算法。
沒有符合。
抽象狀態 $(l_{ERR}, v_\top)$ 違反 φ, 因為其操作為 ERR.
基於此因, φ 違反 DFA 演算法
注意: 這是個錯誤必報

2.
利用模型分析 (model checking) 來檢查 $A \vdash \varphi$:

a) 利用模型分析 Reach $(L,E,\leadsto,e_0)$ 跟前段所學習分析 CPA 到 CFA A。

這就代表:
- $\leadsto = \leadsto_{CPA}$
- $e_0 = (l_0, v_\top)$

請列出回傳抽象集合 reached

$$\text{reached} = \{(l_0,(l_0,v_\top)),\ (l_1,(l_1,v_\top)),\ (l_2,(l_2,v_\top)),\ (l_3,(l_3,\{x\mapsto 0, y\mapsto \top_{\mathbb{Z}}\})),\ (l_3,(l_3,\{x\mapsto -1, y\mapsto \top_{\mathbb{Z}}\})),\ (l_5,(l_5,\{x\mapsto 0, y\mapsto \top_{\mathbb{Z}}\})),\ (l_5,(l_5,\{x\mapsto -1, y\mapsto \top_{\mathbb{Z}}\})),\ (l_f,(l_f,\{x\mapsto 0, y\mapsto \top_{\mathbb{Z}}\})),\ (l_f,(l_f,\{x\mapsto -1, y\mapsto \top_{\mathbb{Z}}\}))\}$$

b) reached 集合裡面的所有抽象狀態都符合 φ 嗎?
請指出是否屬性 φ 符合或違反上述演算法?
符合。
根據 Reach 演算法, φ 滿足, 因為沒有抽象狀態在 l_{ERR} 存在。

演算法 1 資料流演算法(DFA)跟軟體模型檢查法的典型差異 (Reach 演算法)

輸入： 集合：L　　初始抽象狀態：(l_0, e_0)（$l_0 \in L$，$e_0 \in E$，∈ → 屬於）
抽象域：E
轉換關係：$\rightsquigarrow$

輸出： 可抵達的抽象狀態集合（位置跟抽象資料狀態的對）

(a) 演算法 DFA($L, E, \rightsquigarrow, e_0$)

變數：映射：$reached : L \rightarrow E$（→ 抽象映射，表示給定 L 代入時，回傳 E）
集合：$waitlist \subseteq L$

1. $waitlist := \{l_0\}$
2. $reached(l_0) := e_0$（:= 賦值）
3. while $waitlist \neq \{\}$ do
4. 　從 waitlist 取出 l
5. 　從 waitlist 移除 $\{l\}$（→ 不小於等於（偏序集合專用））
6. 　for each (l', e') with $(l, e) \rightsquigarrow (l', e')$ do
7. 　　// 如果仍沒有完全涵蓋
8. 　　如果 $e' \not\sqsubseteq reached(l')$ 則
9. 　　　// 結合(join)既有的抽象資料狀態
10. 　　　$reached(l') := reached(l') \sqcup e'$
11. 　　　$waitlist := waitlist \cup \{l'\}$
12. 回傳 reached

(b) 演算法 Reach($L, E, \rightsquigarrow, e_0$)

變數：集合：$reached \subseteq L \times E$（→ 類似於笛卡兒積的「乘法」）
集合：$waitlist \subseteq L \times E$

1. $waitlist := \{(l_0, e_0)\}$
2. $reached := \{(l_0, e_0)\}$
3. while $waitlist \neq \{\}$ do
4. 　從 waitlist 中取出 (l, e)
5. 　$waitlist := waitlist \setminus \{(l, e)\}$
6. 　for each (l', e') with $(l, e) \rightsquigarrow (l', e')$ do
7. 　　// 如果沒有完全涵蓋
8. 　　如果 $\nexists (l', e'') \in reached : e' \sqsubseteq e''$ 則
9. 　　　// 新增新的抽象狀態
10. 　　　$reached := reached \cup \{(l', e')\}$
11. 　　　$waitlist := waitlist \cup \{(l', e')\}$
12. 回傳 reached

演算法 2 常見條件分析法 CPA($\mathbb{D}, e_0$)

輸入： CPA：$\mathbb{D} = (D, \rightsquigarrow, merge, stop)$　　束：D
初始抽象狀態：e_0　$(e_0 \in E)$
集合：E（束 D 的所有元素集合）

輸出： 可抵達的抽象狀態集合

變數：集合：$reached \subseteq E$
$waitlist \subseteq E$

1. $waitlist := \{e_0\}$
2. $reached := \{e_0\}$
3. while $waitlist \neq \{\}$ do
4. 　從 waitlist 取出 e
5. 　$waitlist := waitlist \setminus \{e\}$
6. 　for each e' with $e \rightsquigarrow e'$ do
7. 　　for each $e'' \in reached$ do
8. 　　　// 跟已知抽象狀態合併
9. 　　　$e_{new} := merge(e', e'')$
10. 　　　如果 $e_{new} \neq e''$ 則
11. 　　　　$waitlist := (waitlist \cup \{e_{new}\}) \setminus \{e''\}$
12. 　　　　$reached := (reached \cup \{e_{new}\}) \setminus \{e''\}$
13. 　　如果 ¬$stop(e', reached)$ 則（否定）
14. 　　　$waitlist := waitlist \cup \{e'\}$
15. 　　　$reached := reached \cup \{e'\}$
16. 回傳 reached