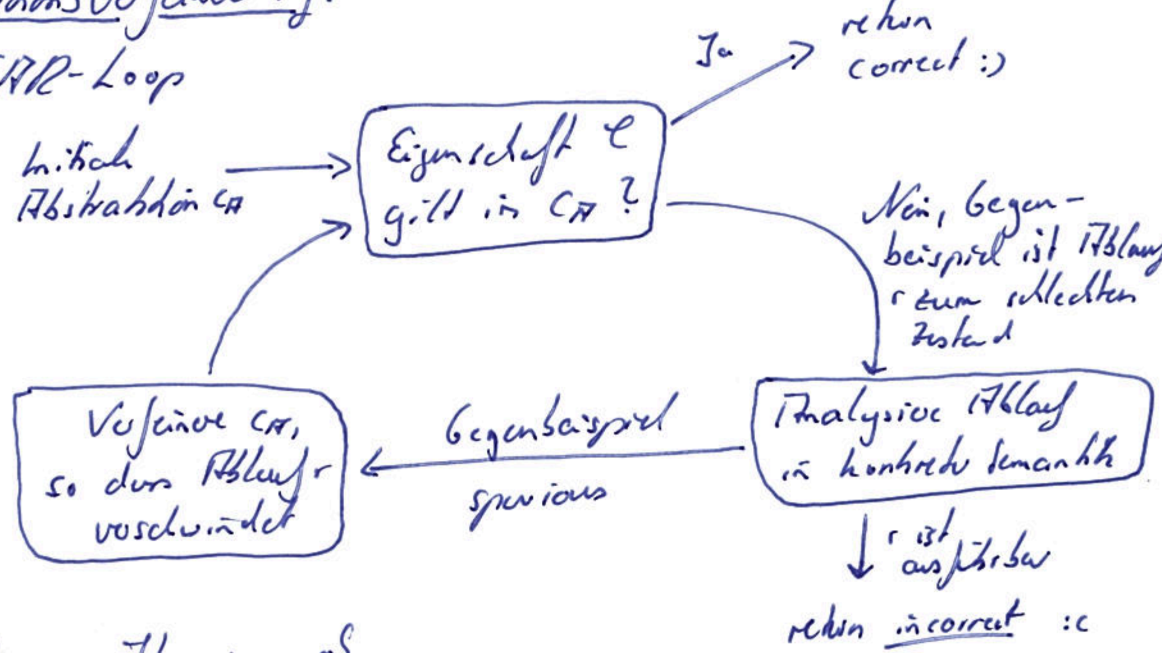
5.3 Abstraktionsverfeinerung:

Ziel: CEGAR-Loop



Probleme: · Wie prüft man, ob Gegenbeispiel echt ist?
· Wie extrahiert man neue Prädikate aus einem spurious Gegenbeispiel?

Typische Eigenschaften C:
- Bestimmte Befehle nicht erreichbar (Dead-Code)
- Keine Division durch 0.
- x wird nie negativ
- Bei Terminierung ist y gerade

Gemeinsamkeit: Eigenschaft lässt sich als Vermeiden einer Konfiguration einem unerwünschten Programm (c, σ) mit $c = c_{bad}$ formulieren.

Definition (Gegenbeispiel):
Betrachte die abstrakte Semantik des Programms c unter Abs(P).
Sei $c_{bad} \in Prog$ ein unerwünschtes Programm.
- Ein Gegenbeispiel ist eine Folge abstrakter Transitionen
  $(c, true) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_n, q_n)$
  mit · $c_n = c_{bad}$ und · $q_i \not\models false$ für alle $1 \leq i \leq n$.
- Das Gegenbeispiel heißt echt, falls es Zustände $\sigma_0, \ldots, \sigma_n \in State$ gibt mit · $\sigma_i \models q_i$ für $1 \leq i \leq n$ und
  $(c, \sigma_0) \rightarrow (c_1, \sigma_1) \rightarrow \ldots \rightarrow (c_n, \sigma_n)$.
- Andernfalls heißt das Gegenbeispiel spurious.

Lemma:

Das Gegenbeispiel $(c_0, true) \Rightarrow \ldots \Rightarrow (c_k, q_k)$ ist spurious gdw. es Prädikate $p_0, \ldots, p_k$ gibt mit

- $p_0 \models\!\!\dashv true$, $p_k \models\!\!\dashv false$ und
- für alle $1 \leq i \leq k$ und alle $\sigma, \sigma' \in State$ mit $\sigma \models p_{i-1}$ und $(c_{i-1}, \sigma) \to (c_i, \sigma')$ gilt:

$$\sigma' \models p_i.$$

Intuition:

- Sei r das azyklische Programm, das den Befehlen der Folge entspricht (dabei werden if b und while b zu assert b / $\neg b$).
- Dann ist r spurious genau dann, wenn

$$\{true\}\ r\ \{false\}$$

ein gültiges Hoare-Tripel ist.

- Das gilt, falls

$true \models\!\!\dashv wp(r, false)$ bzw. $sp(true, r) \models\!\!\dashv false$.

Beweis:

- Definiere p_i als stärkste Nachbedingung, angefangen bei $p_0 := true$.
- Es wird p_i abhängig von p_{i-1} und den Axiomen der Transitionsrelation definiert:

(skip) $p_i := p_{i-1}$

(assign) $(x := a)$ $p_i := \exists x' : (p_{i-1}\{x'/x\} \wedge x = (a\{x'/x\}))$

(while/if-true) $p_i := p_{i-1} \wedge b$

(while/if-false) $p_i := p_{i-1} \wedge \neg b$.

Auch eine Konstruktion über schwächste Vorbedingungen ist möglich. □

Beispiel:

$[x := z]^0$;
$[z := z+1]^1$;
$[y := z]^2$;
if $[y = x]^3$ then
  $[skip]^4$
else
  $[skip]^5$
fi

Eigenschaft: Block 4 ist nicht erreichbar.

Initiale Abstraktion: $P = \emptyset$, also $Abs(P) = \{false, true\}$.

(Spurious) Gegenbeispiel:

$(0, true) \Rightarrow (1, true) \Rightarrow (2, true) \Rightarrow (3, true) \Rightarrow (4, true)$.

Konstruktion der Prädikate im Beweis des obigen Lemmas:

$p_0 := true$

$p_1 := \exists x' : (p_0[x'/x] \wedge x = (z[x'/x]))$

$\models\!| \ \exists x' : (true \wedge x = z) \models\!| \ x = z.$

$p_2 := \exists z' : (p_1[z'/z] \wedge z = (z+1[z'/z]))$

$\models\!| \ \exists z' : (x = z' \wedge z = z'+1).$

$p_3 := \exists y' : (p_2[y'/y] \wedge y = (z[y'/y]))$

$\models\!| \ \exists z' : (x = z' \wedge z = z'+1) \wedge y = z$

$p_4 := p_3 \wedge y = x$

$\models\!| \ \exists z' : (x = z' \wedge z = z'+1) \wedge y = z \wedge y = x$

$\models \ z = x+1 \wedge y = z \wedge y = x$

$\models \ y = x+1 \wedge y = x$

$\models \ false.$

<u>Abstraktionsverfeinerung</u>: · Seien $p_1, \dots, p_{n-1}$ die neu bestimmten Prädikate aus obigem Lemma.

· Definiere $P' := P \cup \{p_1, \dots, p_{n-1}\}$

<u>Lemma:</u>

Berechnet man die abstrakte Semantik von Programm c mit $Abs(P')$, dann gibt es keine Folge

$(c, true) \Rightarrow (c_1, q_1') \Rightarrow \dots \Rightarrow (c_k, q_k')$

mit
- c_i die Programme des vorherigen Gegenbeispiels und
- q_k' ~~⇔~~ false.

<u>Im Beispiel:</u>

$P' = \{\underbrace{x = z}_{=: p_1}, \underbrace{\exists z' : (x = z' \wedge z = z'+1)}_{=: p_2}, \underbrace{\exists z' : (x = z' \wedge z = z'+1) \wedge y = z}_{=: p_3}$

Verfeinere die abstrakte Transitionsrelation:

$$(0, true) \Rightarrow (1, p_1 \wedge \neg p_2 \wedge \neg p_3)$$
$$\Rightarrow (2, \neg p_1 \wedge p_2) \Rightarrow (3, \neg p_1 \wedge p_2 \wedge p_3)$$
$$\Rightarrow (4, \underbrace{\neg p_1 \wedge p_2 \wedge p_3 \wedge x = y}_{\models\dashv\, false})$$

<u>Beweis (des Lemmas):</u>

- Angenommen es gibt eine Folge
  $$(c, true) \Rightarrow (c_1, q_1') \Rightarrow \dots \Rightarrow (c_k, q_k')$$
  mit $q_k' \models\dashv false$.
- Die abstrakte Semantik ist also stärkste Nachbedingungen definiert:
  $$q_{i+1}' := \overline{sp(q_i', c_{i+1})},$$
  wobei c_{i+1} als Befehl aufgefasst wird.
- Da q_1' die stärkste Nachbedingung von true und c_1 ist, die in $Abs(P')$ ausgedrückt werden kann, und da auch p_1 eine solche Nachbedingung ist, folgt
  $$q_1' \models p_1.$$
- Allgemein gilt (das ist eigentlich ein Induktionsschritt, wobei man bereits $q_0' \models p_0$ (beide true) als Basisfall hätte nehmen können):
  Angenommen $q_i' \models p_i$.
  Dann folgt mit Monotonie von $sp(\cdot, c)$:
  $$sp(q_i', c_{i+1}) \models sp(p_i, c_{i+1}) \models p_{i+1}$$
  Mit dem Lemma der letzten Vorlesung hat man:
  $$\overline{sp(q_i', c_{i+1})} \models \overline{p_{i+1}}$$
  Da also $\overline{sp(q_i', c_{i+1})} \models\dashv q_{i+1}'$ und $\overline{p_{i+1}} \models\dashv p_{i+1}$, gilt:
  $$q_{i+1}' \models p_{i+1}.$$
- Also $q_i' \models p_i$ für alle $0 \le i \le k$.
  Da also $p_k \models\dashv false$, muss $q_k' \models\dashv false$ folgen. ↯ □

5.4 Optimierungen

(1) ↳ CEGAR schlägt manchmal fehl:

$[x:=a]^0;$
$[y:=b]^1;$
while $[\neg(x=0)]^2$ **do**
  $[x:=x-1]^3;$
  $[y:=y-1]^4;$
od
if $[a=b \wedge \neg(y=0)]^5$ **then**
  $[skip]^6;$
else
  $[skip]^7;$
fi

- Block 6 nicht erreichbar.
- CEGAR liefert unendliche Folge von Gegenbeispielen mit Prädikaten
  $x=a-k,\ y=b-k$
  für alle $k \in \mathbb{N}$.
- Benötigt wird die Schleifeninvariante
  $a=b \rightarrow x=y$.

↳ Verwandtes Problem:
Prädikate werden unnötig komplex
und beziehen sich auf irrelevante Variablen.

Lösung: Craig-Interpolanten

Definition:
Seien $b, p \in BExp$ mit $b \models p$.
Eine Formel $r \in BExp$ mit
$b \models r$ und $r \models p$ sowie $Vars(r) \subseteq Vars(b) \cap Vars(p)$
heißt Craig-Interpolante.

Satz (Logik):
- Craig-Interpolanten existieren in Aussagenlogik und in Prädikatenlogik erster Stufe.
- Sie lassen sich aus Resolutionsbeweisen ablesen.

(2) Anstatt Prädikate uniform für alle Blöcke zu verwenden, generiere Prädikate gezielt pro Block.
⇒ Lazy abstraction (Ranjit Jhala, UC San Diego),
auch über Craig-Interpolanten.