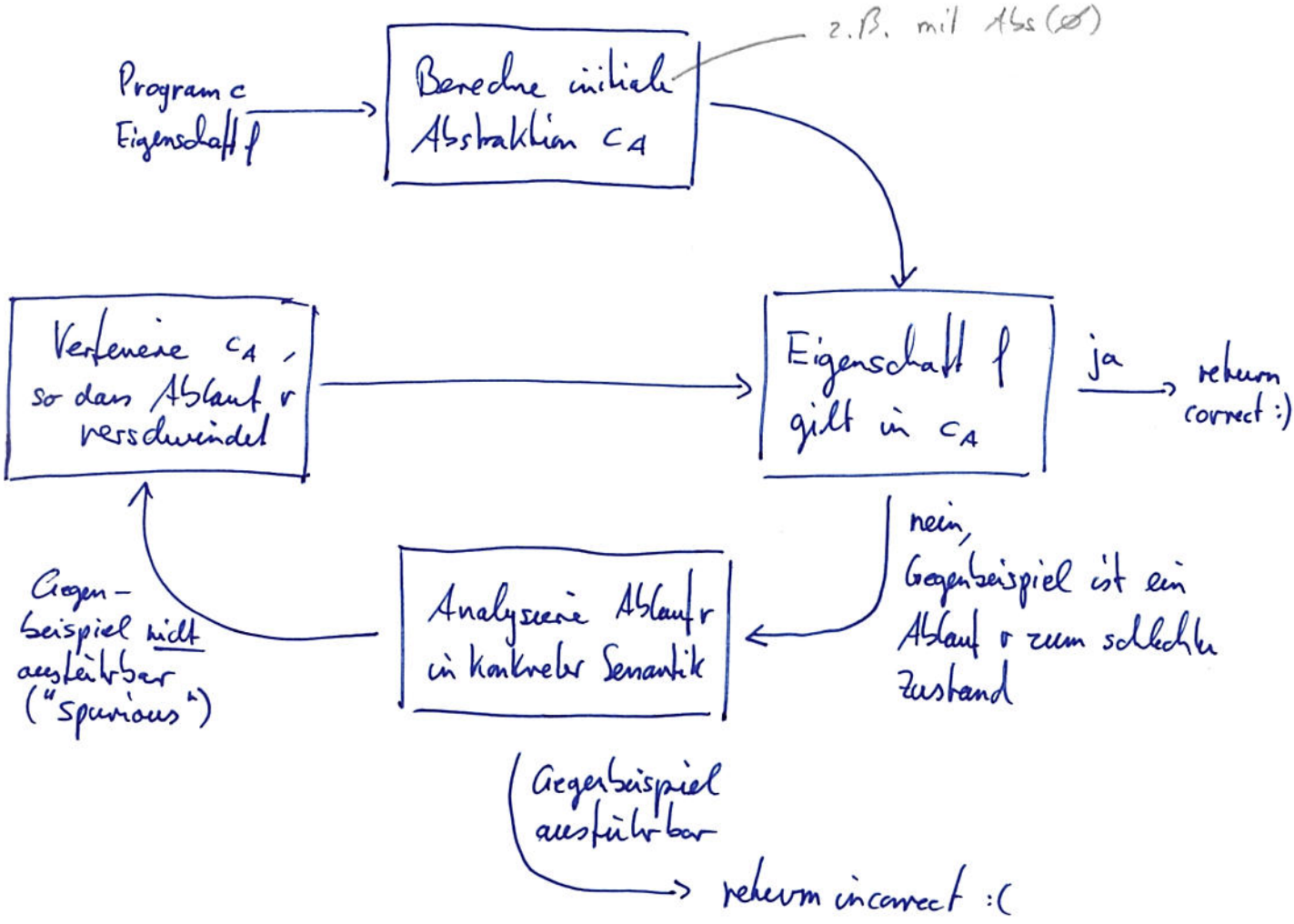
Abstraktionsverfeinerung

Bisher haben wir die Prädikate zur Prädikatenabstraktion als gegeben angesehen.

Jetzt wollen wir diese Prädikate automatisch/iterativ berechnen.

Ziel: CEGAR-Loop

↳ counter-example guided abstraction refinement



Eigenschaften f:

Wir betrachten sog. Safety-Eigenschaften, d.h. Eigenschaften der Form:

Block/Program c ist nicht erreichbar.

D.h. wir wollen Konfiguration (c, σ) vermeiden, mit $c = c_{bad}$.

schlechter/unerwünschter

Definition

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

$$(c, true) \Rightarrow (c_1, q_1) \Rightarrow \dots \Rightarrow (c_K, q_K)$$

mit $c_K = c_{bad}$ und $q_i \not\models false$ für alle $1 \le i \le K$.

- Das Gegenbeispiel heißt *echt*, falls es Zustände $\sigma_0, \dots, \sigma_K \in State$ gibt, mit $\sigma_i \models q_i$ für alle $1 \le i \le K$ und // $\sigma_0 \in \gamma(true)$ $\rightsquigarrow$ also klar.

$\sigma_i \in \gamma(q_i)$

$$(c, \sigma_0) \longrightarrow (c_1, \sigma_1) \longrightarrow \dots \longrightarrow (c_K, \sigma_K)$$

- Andernfalls heißt das Gegenbeispiel *unecht* bzw. *spurious*.

Bemerkung:

Wir identifizieren zu einem Gegenbeispiel

$$(c, true) \Rightarrow (c_1, q_1) \Rightarrow \dots \Rightarrow (c_K, q_K)$$

das azyklische Programm $r = r_1; \dots; r_K$, welches der Folge an Befehlen entspricht die beim Auswerten des Gegenbeispiels abgearbeitet werden.
Wir nennen r einen *Ablauf*. (In der Literatur auch manchmal Line-Program.)
Beachte, dass in r If- und While-Anweisung zu Assume-Statements werden.

Beispiel

Zum Gegenbeispiel

$$(x = x+1;\ \text{if } y \neq 0 \text{ then } x = x-y \text{ else skip end}^{r},\ true) \Rightarrow (\text{if } y \neq 0 \text{ then } x = x-y \text{ else skip end}^{r}, q_1)$$
$$\Rightarrow (x = x-y^{r},\ q_2) \Rightarrow (skip^{r},\ q_3)$$

assoziieren wir den Ablauf

$$r \equiv x = x+1;\ assume(y \neq 0);\ x = x-y$$

Problem: Wie prüfen wir, ob ein Gegenbeispiel *echt* ist?

Lemma:

Sei $c = (c_0, \text{true}) \overset{q_0}{\Rightarrow} \cdots \Rightarrow (c_K, q_K)$ ein Gegenbeispiel.
Ferner sei $r = r_1, \ldots, r_K$ der zum Gegenbeispiel c assoziierte Ablauf.
Dann gilt:

Gegenbeispiel c ist *spurious* gdw. $\not\models \{\text{true}\}\, r\, \{\text{false}\}$

Korollar 1

Sei c ein Gegenbeispiel und $r = r_1, \dots, r_k$ der assoziierte Ab.Conf.
Dann ist c spurious gdw. es Prädikate $p_0, \dots, p_k$ gibt mit

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

} also $\models \{p_{i-1}\}\ r_i\ \{p_i\}$

Beweis: $p_0, \dots, p_k$ existieren gdw. $\{\text{true}\}\ r\ \{\text{false}\}$ gültiges Hoare-Tripel. ∎

Bemerkung: Der Beweis für obiges Lemma benutzt sp.
Wir können aber bel. Prädikate nach Korollar wählen.
Insbesondere können wir wlp nutzen.

Beispiel:

```
[x = z]^0
[z = z+1]^1
[y = z]^2
if [y = x]^3 then
   [skip]^4
else
   [skip]^5
end
```

- Eigenschaft: Block 4 ist nicht erreichbar.
- Initiale Abstraktion: $P = \emptyset$, also
  $Abs(P) = \{\text{false}, \text{true}\}$
- spurious Gegenbeispiel:

$c = (0, \text{true}) \Rightarrow (1, \text{true}) \Rightarrow (2, \text{true}) \Rightarrow (3, \text{true}) \Rightarrow (4, \text{true}$

$\rightsquigarrow$ also $r = 0; 1; 2; 3$

↳ assume

- Zeige, dass c tatsächlich spurious ist.

Darum zeige $\not\vdash \{true\}\ r\ \{false\}$

$\{z+1 \neq z\} \not\Rightarrow \{true\}$

$[x = z]^0$

$\{z+1 \neq x\}$

$[z = z+1]^1$

$\{z \neq x\}$

$[y = z]^2$

$\{y = x \Rightarrow false\} \not\Rightarrow \{y \neq x\}$

assume $[y = x]^3$

$\{false\}$

Hoare-Beweis
→ "rückwärts getrieben"

Also gilt $\not\vdash \{true\}\ r\ \{false\}$.
Somit ist c tatsächlich spurious!

Problem: Wie verfeinert man die Abstraktion?

Abstraktionsverfeinerung:

Sei c ein Gegenbeispiel in $Abs(P)$ und r der darin enthaltene Ablauf.
Wir reichern $Abs(P)$ mit den Prädikaten aus dem obigen Korollar an.
Also: seien $p_0, \dots, p_K$ Prädikate die das obige Korollar liefert.
Dann wähle:

$$P' = P \cup \underbrace{\{p_1, \dots, p_{K-1}\}}_{=: extract(r)}$$

Beobachtung:

Die Prädikate $p_1, \dots, p_{K-1}$ können aus dem Nachweis, dass r spurious ist, extrahiert werden.
Darin nehme z.B. die Assertions aus einen Hoare-Beweis.

Bemerkung:

Man kann $p_1, \dots, p_{K-1}$ auch in die atomaren Bestandteile zerlegen.
Zum Beispiel: $b_1 \wedge b_2, b_3$ wähle statt $b_1 \wedge (b_2 \wedge b_3)$

Beispiel (Fortsetzung):

Wir extrahieren folgende Prädikate aus dem obigen Hoare Beweis:

$$P' = \underbrace{P}_{=\emptyset} \cup \{ \underbrace{z+1 \neq x}_{p_1}, \underbrace{z \neq x}_{p_2}, \underbrace{y \neq x}_{p_3} \}$$

Betrachte erneut $r = 0,1;2;3$.

$$(0, \text{true}) \xrightarrow{[x=z]^0} \Longrightarrow \xrightarrow{[z=z+1]^1} (1, p_1 \wedge \neg p_2)$$

$$\Longrightarrow \xrightarrow{[y=z]^2} (2, p_1 \wedge p_2)$$

$$\Longrightarrow \xrightarrow{\text{assume } [y=x]^3} (3, p_1 \wedge p_2 \wedge p_3)$$

$$\Longrightarrow \overline{p_1 \wedge p_2 \wedge p_3 \wedge \neg p_3}$$

$$\models \text{false}$$

Damit ist Block 4 nicht mehr erreichbar.
Insbesondere ist c (bzw. r) kein Gegenbeispiel mehr. // q_K ist jetzt false

Lemma:

Sei $(c, \text{true}) \Rightarrow (c_1, q_1) \Rightarrow \cdots \Rightarrow (c_K, q_K)$ ein (spurious) Gegenbeispiel von Program c in $Abs_0(P)$. Sei r der dazugehörige Ablauf. Berechnet man die abstrakte Semantik von c mit $Abs(P')$, $P' = P \cup \text{extract}(r)$, dann ex. keine Folge

$$(c, \text{true}) \Rightarrow (c_1, q_1') \Rightarrow \cdots \Rightarrow (c_K, q_K')$$

mit $q_i' \not\models \text{false}$ für alle $1 \leq i \leq K$.

D.h. es ex. kein Gegenbeispiel, dessen assoziierter Ablauf r ist.
$\rightsquigarrow$ "r verschwindet"

Beweis:

Angenommen ex.

$$\underset{\text{c}}{(c_0, q_0)} \Rightarrow \underset{\text{true}}{(c_1, q_1)} \Rightarrow \cdots \Rightarrow (c_K, q_K)$$

mit $q_i' \not\models \text{false}$ für alle $0 \leq i \leq K$. (Insbesondere $q_K' \not\models \text{false}$)

Wie zuvor gilt: $(r_{i+1}, q_i) \Rightarrow q_{i+1}$

Da das Gegenbeispiel spurious ist, ex Prädikate $p_0, \ldots, p_K$, mit

- $\text{extract}(r) = p_1, \ldots, p_{K-1}$
- $p_0 \models \text{true}$, $p_K \not\models \text{false}$
- $\{p_{i-1}\}\, r_i \,\{p_i\}$

Zeigen: $q_i' \models p_i$

IA: $q_0' \models \text{true} \models p_0$

IV: $q_i' \models p_i$

IS: zeige $q_{i+1}' \models p_{i+1}$

- Nach Def sp gilt: $\{A\}\,\tilde{c}\,\{B\}$ impliziert $sp(A, \tilde{c}) \models B$
- Es gilt $\{q_i'\}\, r_{i+1} \,\{p_{i+1}\}$ da nach $\{p_i\}\, r_{i+1} \,\{p_{i+1}\}$ und $q_i' \models p_i$ aus IV mittels (CONS)
- Also $sp(q_i', r_{i+1}) \models p_{i+1}$
- ~~Nun~~ folgt aus Lemma 3.1: $\overline{sp(q_i', r_{i+1})} \models \overline{p_{i+1}}$
- Nach Def $\Rightarrow$ also: $q_{i+1}' \models \overline{p_{i+1}}$
- für $i+1 < K$ gilt: $p_{i+1} \in \text{extract}(r)$ $\rightsquigarrow$ also $\overline{p_{i+1}} \not\models p_{i+1}$
  für $i+1 = K$ gilt: $p_{i+1} \not\models p_K \not\models \text{false}$ $\rightsquigarrow$ $\overline{p_{i+1}} \not\models \text{false} \not\models p_{i+1}$
- In jeden Fall also: $\overline{p_{i+1}} \not\models p_{i+1}$

Damit gilt dann $q_{i+1}' \not\models p_{i+1}$

Jetzt gilt: $q_K' \models p_K \models \text{false}$. Also muss $q_K' \not\models \text{false}$ gelten ↯ ∎

Optimierungen:

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

Beispiel:

ws_0 $\{z+1 \neq a \vee z \neq a\} \overset{0}{\Rightarrow} \{true\}$ $\{true\}$ s_0

$[x = z]$

ws_1 $\{z+1 \neq a \vee x \neq a\}$ $\{\exists x'.\ x = z\} \overset{1}{\Rightarrow} \{x = z\}$ s_1

$[z = z+1]$

ws_2 $\{z \neq a \vee x \neq a\}$ $\{\exists z'.\ x = z' \wedge z = z'+1\} \Rightarrow \{z = x+1\}$ s_2

$[y = z]^2$

ws_3 $\{y \neq a \vee x \neq a\}$ $\{\exists y'.\ z = x+1 \wedge y = z\} \overset{3}{\Rightarrow} \{z = x+1 \wedge y = z\}$ s_3

assume $[x = a]^3$

ws_4 $\{y \neq a\}$ $\{z = x+1 \wedge y = z \wedge x = a\}$ s_4

assume $[y = a]^4$

ws_5 $\{false\}$ $\{z = x+1 \wedge y = z \wedge x = a \wedge y = a\} \Rightarrow \{false\}$ s_5

Lösung: Craig-Interpolation zur Vereinfachung von neuen Prädikaten.

Definition:

Seien $b, p \in BExp$ mit $b \models p$.
Eine Formel $r \in BExp$ mit

- $b \models r$
- $r \models p$
- $Vars(r) \subseteq Vars(b) \cap Vars(p)$

heißt **Craig-Interpolante**.

Satz (Logik):

- Craig-Interpolanten existieren in Aussagenlogik und Prädikatenlogik erster Stufe
- Sie lassen sich aus Resolutionsbeweisen ablesen

Anwendung:

- Berechne für spurious Ablauf $r = r_1; \ldots, r_n$:

  $s_i = sp(true, r_1; \ldots; r_i)$

  $w_i = wp(r_{i+1}; \ldots; r_n, false)$

- Wähle als neue Prädikate

  c_i mit c_i Craig Interpolante von s_i und w_i

  ↳ HA: zeige $s_i \models w_i$ gilt

- Überprüfe sog. Tracking Property:

  Es muss gelten, dass folgende Hoare Tripel gültig sind:

  $\{true\}\, r_1\, \{c_1\},\ \{c_1\}\, r_2\, \{c_2\},\ \ldots,\ \{c_{n-2}\}\, r_{n-1}\, \{c_{n-1}\},\ \{c_{n-1}\}\, r_n\, \{false\}$

  ⇝ Dies gilt z.B., falls man immer die schwächste/stärkste Craig Interpolante wählt.

Beispiel: (Fortsetzung)

i	s_i	w_i	stärkste c_i	schwächste c_i'
1	$x = z$	$z+1 \neq a \vee x \neq a$ $\models z+1 \neq a$	$x = z$	$x \leq z$
2	$z = x+1$	$z \neq a \vee x \neq a$ $\models z \neq a$	$z = x+1$	$x < z$
3	$z = x+1 \wedge y = z$ $\models y = x+1$	$y \neq a \vee x \neq a$ $\models y \neq x$	$y = x+1$	$x < y$
4	$z = x+1 \wedge y = z$ $\wedge\ x = a$ $\models y = a+1$	$y \neq a$	$y = a+1$	$y < a$

Man kann zeigen, dass die Tracking-Property für obige Interpolanten gilt.

Bemerkung:

CEGAR kann dennoch terminieren.
↳ Prädikate können unendlich lange verfeinert werden,
ohne dass die gewünschte Eigenschaft gezeigt werden kann.

Beispiel:

```
[x = a]^0;
[y = b]^1;
while [x ≠ 0]^2 do
    [x = x - 1]^3;
    [y = y - 1]^4;
end
if [a = b ∧ y ≠ 0] then
    [skip]^6;
else
    [skip]^7;
end
```

- Block 6 nicht erreichbar
- CEGAR leitet unendliche Folge an Gegenbeispielen mit Prädikaten

  ~~x = a~~ $x = k$, $y = k$ für alle $k \in \mathbb{N}$

  (bzw in sp Fall: $x = a - k$, $y = b - k$)

- Benötigt wird die Schleifeninvariante

  $a = b \rightarrow x = y$