

Interprétation abstraite et analyse du flot de données

pour l'analyse complètement automatique de programmes

Initiée par G. Kildall en 1972,
généralisée par P. et R. Cousot en 1977 à **Grenoble !**

Interprétation abstraite et analyse du flot de données

pour l'analyse complètement automatique de programmes

Initiée par G. Kildall en 1972,
généralisée par P. et R. Cousot en 1977 à **Grenoble !**

Qu'est-ce ? méthode pour **inférer automatiquement** des propriétés d'une **certaine forme** sur les programmes (e.g. invariants de boucle).

Interprétation abstraite et analyse du flot de données

pour l'analyse complètement automatique de programmes

Initiée par G. Kildall en 1972,
généralisée par P. et R. Cousot en 1977 à **Grenoble !**

Qu'est-ce ? méthode pour **inférer automatiquement** des propriétés d'une **certaine forme** sur les programmes (e.g. invariants de boucle).

Pour ? optimiser les programmes (compilation) ;
garantir l'absence de RTE dans systèmes critiques (avions, nucléaire, ferroviaire, etc) ;
détecter vulnérabilités potentielles (sécurité du code).

Interprétation abstraite et analyse du flot de données

pour l'analyse complètement automatique de programmes

Initiée par G. Kildall en 1972,
généralisée par P. et R. Cousot en 1977 à **Grenoble !**

Qu'est-ce ? méthode pour **inférer automatiquement** des propriétés d'une **certaine forme** sur les programmes (e.g. invariants de boucle).

Pour ? optimiser les programmes (compilation) ;
garantir l'absence de RTE dans systèmes critiques (avions, nucléaire, ferroviaire, etc) ;
détecter vulnérabilités potentielles (sécurité du code).

Comment ? en approximant les déductions de la logique de Hoare (ou des variantes/extensions).

Plan du Cours 3

Idées de base

Interpréteur “en avant” intra-procédural générique et correct

Théorie des *treillis bornés* et applications aux domaines abstraits

Analyses de valeurs et applications

Analyses des variables vivantes et applications

Combiner analyses du “flot de donnée” et du “flot de contrôle”

Approximer les réponses pour terminer (rapidement)

Soit Ω une classe de programmes, e.g. “programmes S sans RTE”.
Réponse oui/non à question “ $S \in \Omega$?” indécidable (thm de Rice).

\rightsquigarrow approximée par procédure de décision $\pi(S)$
terminant et donnant une réponse parmi {oui, non, NSPP}.

Approximer les réponses pour terminer (rapidement)

Soit Ω une classe de programmes, e.g. “programmes S sans RTE”.
Réponse oui/non à question “ $S \in \Omega$?” indécidable (thm de Rice).

\rightsquigarrow approximée par procédure de décision $\pi(S)$
terminant et donnant une réponse parmi {oui, non, NSPP}.

Soit $\pi_1 = \{S \mid \pi(S) = \text{oui}\}$ et $\pi_0 = \{S \mid \pi(S) = \text{non}\}$

méthode correcte $\pi_1 \subseteq \Omega$ et $\pi_0 \subseteq \bar{\Omega}$

méthode complète $\Omega \subseteq \pi_1$

bug finding $\pi_0 \subseteq \bar{\Omega}$

fausses alarmes $\Omega \setminus \pi_1$ (= faux positifs)

bugs manqués $\pi_1 \setminus \Omega$ (= faux négatifs)

Approximer les réponses pour terminer (rapidement)

Soit Ω une classe de programmes, e.g. “programmes S sans RTE”.
Réponse oui/non à question “ $S \in \Omega$?” indécidable (thm de Rice).

\rightsquigarrow approximée par procédure de décision $\pi(S)$
terminant et donnant une réponse parmi {oui, non, NSPP}.

Soit $\pi_1 = \{S \mid \pi(S) = \text{oui}\}$ et $\pi_0 = \{S \mid \pi(S) = \text{non}\}$

méthode correcte $\pi_1 \subseteq \Omega$ et $\pi_0 \subseteq \bar{\Omega}$

méthode complète $\Omega \subseteq \pi_1$

bug finding $\pi_0 \subseteq \bar{\Omega}$

fausses alarmes $\Omega \setminus \pi_1$ (= faux positifs)

bugs manqués $\pi_1 \setminus \Omega$ (= faux négatifs)

Pour optimisation ou sûreté critique : méthodes **correctes**.

La sécu se contente d'un “bug finding”, ni correct, ni complet.

Conditions logiques comme sous-ensemble d'états

Condition “ $x^2 + y^2 \leq 42$ ” représente

$$\{ m \in \mathbb{M} \mid m(x)^2 + m(y)^2 \leq 42 \}$$

Conditions logiques comme sous-ensemble d'états

Condition " $x^2 + y^2 \leq 42$ " représente

$$\{ m \in \mathbb{M} \mid m(x)^2 + m(y)^2 \leq 42 \}$$

Réciproquement, soit A un sous-ensemble d'états,

on note $\gamma(A)$ sa *propriété caractéristique* (ou **concrétisation**).

Conditions logiques comme sous-ensemble d'états

Condition “ $x^2 + y^2 \leq 42$ ” représente
 $\{ m \in \mathbb{M} \mid m(x)^2 + m(y)^2 \leq 42 \}$

Réciproquement, soit A un sous-ensemble d'états,
on note $\gamma(A)$ sa *propriété caractéristique* (ou **concrétisation**).

On a la correspondance suivante :

- ▶ $\gamma(A_1 \cup A_2)$ équivaut à $\gamma(A_1) \vee \gamma(A_2)$.
- ▶ $\gamma(A_1 \cap A_2)$ équivaut à $\gamma(A_1) \wedge \gamma(A_2)$.
- ▶ $\gamma(\overline{A_1})$ équivaut à $\neg\gamma(A_1)$.
- ▶ $\gamma(\emptyset)$ équivaut à `false`.
- ▶ $A_1 \subseteq A_2$ équivaut à “ $A_1 \Rightarrow A_2$ valide”
(i.e. implication vraie dans toute interprétation).

Conditions logiques comme sous-ensemble d'états

Condition “ $x^2 + y^2 \leq 42$ ” représente
 $\{ m \in \mathbb{M} \mid m(x)^2 + m(y)^2 \leq 42 \}$

Réciproquement, soit A un sous-ensemble d'états,
on note $\gamma(A)$ sa *propriété caractéristique* (ou **concrétisation**).

On a la correspondance suivante :

- ▶ $\gamma(A_1 \cup A_2)$ équivaut à $\gamma(A_1) \vee \gamma(A_2)$.
- ▶ $\gamma(A_1 \cap A_2)$ équivaut à $\gamma(A_1) \wedge \gamma(A_2)$.
- ▶ $\gamma(\overline{A_1})$ équivaut à $\neg\gamma(A_1)$.
- ▶ $\gamma(\emptyset)$ équivaut à `false`.
- ▶ $A_1 \subseteq A_2$ équivaut à “ $A_1 \Rightarrow A_2$ valide”
(i.e. implication vraie dans toute interprétation).

\rightsquigarrow approximer déductions logiques grâce à
calculs restreints à des classes *limitées* de sous-ensembles
avec test d'inclusion (ie conséquence logique) décidable

Qq classes de “sous-ensembles numériques *convexes*”

sur-approximations *convexes* \sqcup du \vee de conjonctions de contraintes numériques

Conjonctions d'intervalles i.e. *hyper-rectangles*

Exemple pour P, P' deux hyper-rectangles tels que

$$\gamma(P) \equiv -5 \leq x_1 < 5 \wedge -5 < x_2 \leq 5 ; \gamma(P') \equiv -10 \leq x_1 \leq 0 \wedge -10 \leq x_2 \leq 0$$

Enveloppe hyper-rect. $\gamma(P \sqcup_H P') \equiv -10 \leq x_1 < 5 \wedge -10 \leq x_2 \leq 5$

Qq classes de “sous-ensembles numériques *convexes*”

sur-approximations *convexes* \sqcup du \vee de conjonctions de contraintes numériques

Conjonctions d'intervalles i.e. *hyper-rectangles*

Exemple pour P, P' deux hyper-rectangles tels que

$$\gamma(P) \equiv -5 \leq x_1 < 5 \wedge -5 < x_2 \leq 5 ; \gamma(P') \equiv -10 \leq x_1 \leq 0 \wedge -10 \leq x_2 \leq 0$$

Enveloppe hyper-rect. $\gamma(P \sqcup_H P') \equiv -10 \leq x_1 < 5 \wedge -10 \leq x_2 \leq 5$

Conjonctions d'inégalités affines i.e. *polyèdres* (*polytopes*) *convexes*

Exemples avec P, P' ci-dessus

$$\gamma(P \sqcup_C P') \equiv P \sqcup_H P' \wedge -10 < x_2 - x_1 \leq 10$$

Qq classes de “sous-ensembles numériques *convexes*”

sur-approximations *convexes* \sqcup du \vee de conjonctions de contraintes numériques

Conjonctions d'intervalles i.e. *hyper-rectangles*

Exemple pour P , P' deux hyper-rectangles tels que

$$\gamma(P) \equiv -5 \leq x_1 < 5 \wedge -5 < x_2 \leq 5 ; \gamma(P') \equiv -10 \leq x_1 \leq 0 \wedge -10 \leq x_2 \leq 0$$

Enveloppe hyper-rect. $\gamma(P \sqcup_H P') \equiv -10 \leq x_1 < 5 \wedge -10 \leq x_2 \leq 5$

Conjonctions d'inégalités affines i.e. *polyèdres* (*polytopes*) *convexes*

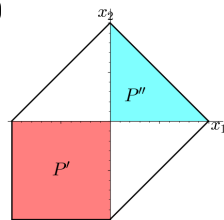
Exemples avec P , P' ci-dessus et P'' ci-dessous

$$\gamma(P \sqcup_C P') \equiv P \sqcup_H P' \wedge -10 < x_2 - x_1 \leq 10$$

Pour P'' (non rectangulaire) tq

$$\gamma(P'') \equiv x_1 + x_2 \leq 10 \wedge 0 \leq x_1 \leq 10 \wedge 0 \leq x_2$$

$$\begin{aligned} \gamma(P' \sqcup_C P'') \equiv & x_1 + x_2 \leq 10 \wedge -10 \leq x_2 - x_1 \leq 10 \\ & \wedge -10 \leq x_1 \wedge -10 \leq x_2 \end{aligned}$$



Qq classes de “sous-ensembles numériques *convexes*”

sur-approximations *convexes* \sqcup du \vee de conjonctions de contraintes numériques

Conjonctions d'intervalles i.e. *hyper-rectangles*

Exemple pour P , P' deux hyper-rectangles tels que

$$\gamma(P) \equiv -5 \leq x_1 < 5 \wedge -5 < x_2 \leq 5 ; \gamma(P') \equiv -10 \leq x_1 \leq 0 \wedge -10 \leq x_2 \leq 0$$

Enveloppe hyper-rect. $\gamma(P \sqcup_H P') \equiv -10 \leq x_1 < 5 \wedge -10 \leq x_2 \leq 5$

Conjonctions d'inégalités affines i.e. *polyèdres* (*polytopes*) *convexes*

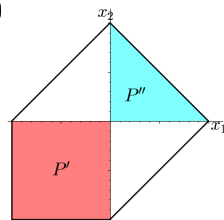
Exemples avec P , P' ci-dessus et P'' ci-dessous

$$\gamma(P \sqcup_C P') \equiv P \sqcup_H P' \wedge -10 < x_2 - x_1 \leq 10$$

Pour P'' (non rectangulaire) tq

$$\gamma(P'') \equiv x_1 + x_2 \leq 10 \wedge 0 \leq x_1 \leq 10 \wedge 0 \leq x_2$$

$$\begin{aligned} \gamma(P' \sqcup_C P'') \equiv & x_1 + x_2 \leq 10 \wedge -10 \leq x_2 - x_1 \leq 10 \\ & \wedge -10 \leq x_1 \wedge -10 \leq x_2 \end{aligned}$$



Une \sqcup convexe vérifie : $\gamma(A_1) \vee \gamma(A_2) \Rightarrow \gamma(A_1 \sqcup A_2)$

Plan du Cours 3

Idées de base

Interpréteur “en avant” intra-procédural générique et correct

Théorie des *treillis bornés* et applications aux domaines abstraits

Analyses de valeurs et applications

Analyses des variables vivantes et applications

Combiner analyses du “flot de donnée” et du “flot de contrôle”

Principe d'un interpréteur abstrait générique correct

Paramétré par un “**domaine abstrait**” \mathbb{D} où

- ▶ \mathbb{D} est un sous-ensemble des sous-ensembles d'états ;
- ▶ l'ensemble \top de tous les états vérifie $\top \in \mathbb{D}$, et donc $\gamma(\top) \equiv \text{true}$;
- ▶ \mathbb{D} est clos pour d'autres opérateurs (cf. diapo 9) ;
- ▶ éléments de \mathbb{D} généralement appelés **états abstraits** et notés A dans ce cours.

Principe d'un interpréteur abstrait générique correct

Paramétré par un “**domaine abstrait**” \mathbb{D} où

- ▶ \mathbb{D} est un sous-ensemble des sous-ensembles d'états ;
- ▶ l'ensemble \top de tous les états vérifie $\top \in \mathbb{D}$, et donc $\gamma(\top) \equiv \text{true}$;
- ▶ \mathbb{D} est clos pour d'autres opérateurs (cf. diapo 9) ;
- ▶ éléments de \mathbb{D} généralement appelés **états abstraits** et notés A dans ce cours.

Décrit par système d'inférence $\#\{A\} S \{A'\}$ tq

$$\#\{A\} S \{A'\} \Rightarrow \{\gamma(A)\} S \{\gamma(A')\}$$

Principe d'un interpréteur abstrait générique correct

Paramétré par un “**domaine abstrait**” \mathbb{D} où

- ▶ \mathbb{D} est un sous-ensemble des sous-ensembles d'états ;
- ▶ l'ensemble \top de tous les états vérifie $\top \in \mathbb{D}$, et donc $\gamma(\top) \equiv \text{true}$;
- ▶ \mathbb{D} est clos pour d'autres opérateurs (cf. diapo 9) ;
- ▶ éléments de \mathbb{D} généralement appelés **états abstraits** et notés A dans ce cours.

Décrit par système d'inférence $\# \{A\} S \{A'\}$ tq

$$\# \{A\} S \{A'\} \Rightarrow \{\gamma(A)\} S \{\gamma(A')\}$$

En analyse arrière, $\# \{A'\} S \{A\}$ *sous*-approxime

calcul de WP $\{P\} S \{\gamma(A)\}$ avec $\gamma(A') \Rightarrow P$

de sorte que $\# \{A\} S \{\top\}$ garantit $\{\gamma(A)\} S \{\text{true}\}$.

Principe d'un interpréteur abstrait générique correct

Paramétré par un “**domaine abstrait**” \mathbb{D} où

- ▶ \mathbb{D} est un sous-ensemble des sous-ensembles d'états ;
- ▶ l'ensemble \top de tous les états vérifie $\top \in \mathbb{D}$, et donc $\gamma(\top) \equiv \text{true}$;
- ▶ \mathbb{D} est clos pour d'autres opérateurs (cf. diapo 9) ;
- ▶ éléments de \mathbb{D} généralement appelés **états abstraits** et notés A dans ce cours.

Décrit par système d'inférence $\# \{A\} S \{A'\}$ tq

$$\# \{A\} S \{A'\} \Rightarrow \{\gamma(A)\} S \{\gamma(A')\}$$

En analyse arrière, $\# \{A'\} S \{A\}$ *sous*-approxime

calcul de WP $\{P\} S \{\gamma(A)\}$ avec $\gamma(A') \Rightarrow P$

de sorte que $\# \{A\} S \{\top\}$ garantit $\{\gamma(A)\} S \{\text{true}\}$.

En analyse avant, $\# \{A\} S \{A'\}$ *sur*-approxime

calcul de *plus forte postcondition* $\{\gamma(A)\} S \{Q\}$ avec $Q \Rightarrow \gamma(A')$

de sorte que $\# \{\top\} S \{A\}$ garantit $\{\text{true}\} S \{\gamma(A)\}$.

Règles *dérivées* du calcul de plus forte post-condition (SP)

$$\frac{}{\{P\} \mathbf{abort} \{ \mathbf{false} \}} \quad P \Rightarrow \mathbf{false}$$

$$\frac{}{\{P\} \mathbf{assume} C \{P \wedge C\}}$$

$$\frac{}{\{P\} x := T \{ \exists x_0, P[x \leftarrow x_0] \wedge x = T[x \leftarrow x_0] \}} \quad x_0 \text{ "frais"}$$

$$\frac{\{P\} S_1 \{I\} \quad \{I\} S_2 \{Q\}}{\{P\} S_1; S_2 \{Q\}}$$

$$\frac{\{P\} S_1 \{Q_1\} \quad \{P\} S_2 \{Q_2\}}{\{P\} S_1 \sqcup S_2 \{Q_1 \vee Q_2\}}$$

$$\frac{\{I\} S \{Q\}}{\{P\} S^* \{I\}} \quad \begin{array}{l} P \Rightarrow I \\ Q \Rightarrow I \end{array}$$

Règles *dérivées* du calcul de plus forte post-condition (SP)

$$\frac{}{\{P\} \mathbf{abort} \{ \mathbf{false} \}} \quad P \Rightarrow \mathbf{false} \qquad \frac{}{\{P\} \mathbf{assume} C \{P \wedge C\}}$$

$$\frac{}{\{P\} x := T \{ \exists x_0, P[x \leftarrow x_0] \wedge x = T[x \leftarrow x_0] \}} \quad x_0 \text{ "frais"}$$

$$\frac{\{P\} S_1 \{I\} \quad \{I\} S_2 \{Q\}}{\{P\} S_1; S_2 \{Q\}} \qquad \frac{\{P\} S_1 \{Q_1\} \quad \{P\} S_2 \{Q_2\}}{\{P\} S_1 \sqcup S_2 \{Q_1 \vee Q_2\}}$$

$$\frac{\{I\} S \{Q\}}{\{P\} S^* \{I\}} \quad P \Rightarrow I \quad Q \Rightarrow I$$

- ▶ génération d'OP uniquement dans **abort** et S^* .
- ▶ opérateur "**var** $x(S)$ " éliminable quitte à renommer avec variables de programmes fraîches.

Interface d'un domaine abstrait \mathbb{D} “en avant”, doit fournir

test d'inclusion \sqsubseteq décidable tq $A \sqsubseteq A'$ garantit $\gamma(A) \Rightarrow \gamma(A')$

Interface d'un domaine abstrait \mathbb{D} “en avant”, doit fournir

test d'inclusion \sqsubseteq décidable tq $A \sqsubseteq A'$ garantit $\gamma(A) \Rightarrow \gamma(A')$

Les opérateurs internes suivants :

top \top tq $\gamma(\top) \equiv \text{true}$

bottom \perp tq $\gamma(\perp) \equiv \text{false}$

Interface d'un domaine abstrait \mathbb{D} “en avant”, doit fournir

test d'inclusion \sqsubseteq décidable tq $A \sqsubseteq A'$ garantit $\gamma(A) \Rightarrow \gamma(A')$

Les opérateurs internes suivants :

top \top tq $\gamma(\top) \equiv \text{true}$

bottom \perp tq $\gamma(\perp) \equiv \text{false}$

garde $A \sqcap C$ tq $(\gamma(A) \wedge C) \Rightarrow \gamma(A \sqcap C)$

Interface d'un domaine abstrait \mathbb{D} “en avant”, doit fournir

test d'inclusion \sqsubseteq décidable tq $A \sqsubseteq A'$ garantit $\gamma(A) \Rightarrow \gamma(A')$

Les opérateurs internes suivants :

top \top tq $\gamma(\top) \equiv \text{true}$

bottom \perp tq $\gamma(\perp) \equiv \text{false}$

garde $A \sqcap C$ tq $(\gamma(A) \wedge C) \Rightarrow \gamma(A \sqcap C)$

affectation $A[x := T]$ tq pour x_0 “frais” dans $\gamma(A)$ et T
 $(\exists x_0, \gamma(A)[x \leftarrow x_0] \wedge x = T[x \leftarrow x_0]) \Rightarrow \gamma(A[x := T])$

Interface d'un domaine abstrait \mathbb{D} “en avant”, doit fournir

test d'inclusion \sqsubseteq décidable tq $A \sqsubseteq A'$ garantit $\gamma(A) \Rightarrow \gamma(A')$

Les opérateurs internes suivants :

top \top tq $\gamma(\top) \equiv \text{true}$

bottom \perp tq $\gamma(\perp) \equiv \text{false}$

garde $A \sqcap C$ tq $(\gamma(A) \wedge C) \Rightarrow \gamma(A \sqcap C)$

affectation $A[x := T]$ tq pour x_0 “frais” dans $\gamma(A)$ et T
 $(\exists x_0, \gamma(A)[x \leftarrow x_0] \wedge x = T[x \leftarrow x_0]) \Rightarrow \gamma(A[x := T])$

join $A_1 \sqcup A_2$ tq $(\gamma(A_1) \vee \gamma(A_2)) \Rightarrow \gamma(A_1 \sqcup A_2)$

Interface d'un domaine abstrait \mathbb{D} “en avant”, doit fournir

test d'inclusion \sqsubseteq décidable tq $A \sqsubseteq A'$ garantit $\gamma(A) \Rightarrow \gamma(A')$

Les opérateurs internes suivants :

top \top tq $\gamma(\top) \equiv \text{true}$

bottom \perp tq $\gamma(\perp) \equiv \text{false}$

garde $A \sqcap C$ tq $(\gamma(A) \wedge C) \Rightarrow \gamma(A \sqcap C)$

affectation $A[x := T]$ tq pour x_0 “frais” dans $\gamma(A)$ et T
 $(\exists x_0, \gamma(A)[x \leftarrow x_0] \wedge x = T[x \leftarrow x_0]) \Rightarrow \gamma(A[x := T])$

join $A_1 \sqcup A_2$ tq $(\gamma(A_1) \vee \gamma(A_2)) \Rightarrow \gamma(A_1 \sqcup A_2)$

widening $A_1 \nabla_g A_2$ tq $(\gamma(A_1) \vee \gamma(A_2)) \Rightarrow \gamma(A_1 \nabla_g A_2)$

∇_g sert à inférer un invariant de boucle par itérations successives de l'interpréteur (cf. diapo suivante).

g = “guide heuristique” (e.g. nombre d'itérations)

pour contrôler convergence (e.g. en retournant \top)

Règles *de base* d'un interpréteur abstrait "en avant"

pour analyse intra-procédurale (i.e. avec inlining des appels ou abstraction brutale)

$$\frac{}{\#\{A\} \mathbf{abort} \{\perp\}} \quad A \sqsubseteq \perp$$

$$\frac{}{\#\{A\} \mathbf{assume} C \{A \sqcap C\}}$$

$$\frac{}{\#\{A\} x := T \{A[x := T]\}}$$

$$\frac{\#\{A\} S_1 \{A_1\} \quad \#\{A_1\} S_2 \{A_2\}}{\#\{A\} S_1; S_2 \{A_2\}}$$

$$\frac{\#\{A\} S_1 \{A_1\} \quad \#\{A\} S_2 \{A_2\}}{\#\{A\} S_1 \sqcup S_2 \{A_1 \sqcup A_2\}}$$

Règles de base d'un interpréteur abstrait "en avant"

pour analyse intra-procédurale (i.e. avec inlining des appels ou abstraction brutale)

$$\frac{}{\#\{A\} \mathbf{abort} \{\perp\}} \quad A \sqsubseteq \perp$$

$$\frac{}{\#\{A\} \mathbf{assume} C \{A \sqcap C\}}$$

$$\frac{}{\#\{A\} x := T \{A[x := T]\}}$$

$$\frac{\#\{A\} S_1 \{A_1\} \quad \#\{A_1\} S_2 \{A_2\}}{\#\{A\} S_1; S_2 \{A_2\}}$$

$$\frac{\#\{A\} S_1 \{A_1\} \quad \#\{A\} S_2 \{A_2\}}{\#\{A\} S_1 \sqcup S_2 \{A_1 \sqcup A_2\}}$$

$$\frac{\#\{A\} S \{A'\}}{\#\{A\} S^* \{A\}} \quad A' \sqsubseteq A$$

$$\frac{\#\{A\} S \{A'\} \quad \#\{A \nabla_g A'\} S^* \{A_f\}}{\#\{A\} S^* \{A_f\}} \quad g \text{ qcq}$$

Règles de base d'un interpréteur abstrait "en avant"

pour analyse intra-procédurale (i.e. avec inlining des appels ou abstraction brutale)

$$\frac{}{\#\{A\} \mathbf{abort} \{\perp\}} \quad A \sqsubseteq \perp$$

$$\frac{}{\#\{A\} \mathbf{assume} C \{A \sqcap C\}}$$

$$\frac{}{\#\{A\} x := T \{A[x := T]\}}$$

$$\frac{\#\{A\} S_1 \{A_1\} \quad \#\{A_1\} S_2 \{A_2\}}{\#\{A\} S_1; S_2 \{A_2\}}$$

$$\frac{\#\{A\} S_1 \{A_1\} \quad \#\{A\} S_2 \{A_2\}}{\#\{A\} S_1 \sqcup S_2 \{A_1 \sqcup A_2\}}$$

$$\frac{\#\{A\} S \{A'\}}{\#\{A\} S^* \{A\}} \quad A' \sqsubseteq A$$

$$\frac{\#\{A\} S \{A'\} \quad \#\{A \nabla_g A'\} S^* \{A_f\}}{\#\{A\} S^* \{A_f\}} \quad g \text{ qcq}$$

Thm de correction $\#\{A\} S \{A'\} \Rightarrow \{\gamma(A)\} S \{\gamma(A')\}$

Analyse des **assert** et applications

Pour plus de précision, on considère la def. alternative suivante

assert $C \stackrel{def}{=} \text{if } C \text{ then } \varepsilon \text{ else abort}$

qui permet de prendre l'interprétation alternative

$$\frac{\# \{A\} \text{ assert } C \{A \sqcap C\}}{A \sqcap \neg C \sqsubseteq \perp}$$

Analyse des **assert** et applications

Pour plus de précision, on considère la def. alternative suivante

assert $C \stackrel{def}{=} \text{if } C \text{ then } \varepsilon \text{ else abort}$

qui permet de prendre l'interprétation alternative

$$\frac{\# \{A\} \text{ assert } C \{A \sqcap C\}}{A \sqcap \neg C \sqsubseteq \perp}$$

En **vérification**, on collecte cette OP pour colorier :

gris si $A \sqsubseteq \perp$ (code garanti mort) ;

vert sinon et si $A \sqcap \neg C \sqsubseteq \perp$ (garanti sans erreur) ;

rouge sinon et si $A \sqcap C \sqsubseteq \perp$ (erreur certaine) ;

orange sinon (on ne sait rien : alarme !)

Analyse des **assert** et applications

Pour plus de précision, on considère la def. alternative suivante

assert $C \stackrel{def}{=} \text{if } C \text{ then } \varepsilon \text{ else abort}$

qui permet de prendre l'interprétation alternative

$$\frac{}{\#\{A\} \text{ assert } C \{A \sqcap C\}} A \sqcap \neg C \sqsubseteq \perp$$

En **vérification**, on collecte cette OP pour colorier :

gris si $A \sqsubseteq \perp$ (code garanti mort) ;

vert sinon et si $A \sqcap \neg C \sqsubseteq \perp$ (garanti sans erreur) ;

rouge sinon et si $A \sqcap C \sqsubseteq \perp$ (erreur certaine) ;

orange sinon (on ne sait rien : alarme !)

En **optimisation** (e.g. des programmes C), **OP ignorées**[†] :

UB absents d'après la précondition du compilateur

NB revient à assimiler **abort** avec \emptyset et **assert** avec **assume**

Analyse des **assert** et applications

Pour plus de précision, on considère la def. alternative suivante

assert $C \stackrel{def}{=} \text{if } C \text{ then } \varepsilon \text{ else abort}$

qui permet de prendre l'interprétation alternative

$$\frac{}{\# \{A\} \text{ assert } C \{A \sqcap C\}} A \sqcap \neg C \sqsubseteq \perp$$

En **vérification**, on collecte cette OP pour colorier :

gris si $A \sqsubseteq \perp$ (code garanti mort) ;

vert sinon et si $A \sqcap \neg C \sqsubseteq \perp$ (garanti sans erreur) ;

rouge sinon et si $A \sqcap C \sqsubseteq \perp$ (erreur certaine) ;

orange sinon (on ne sait rien : alarme !)

En **optimisation** (e.g. des programmes C), **OP ignorées**[†] :

UB absents d'après la précondition du compilateur

NB revient à assimiler **abort** avec \emptyset et **assert** avec **assume**

[†] sauf dans le cas **rouge**, *warning* emis si activé par l'utilisateur !

Plan du Cours 3

Idées de base

Interpréteur “en avant” intra-procédural générique et correct

Théorie des *treillis bornés* et applications aux domaines abstraits

Analyses de valeurs et applications

Analyses des variables vivantes et applications

Combiner analyses du “flot de donnée” et du “flot de contrôle”

Ordre partiel d'un ensemble de parties

Def. Soit \mathbb{D} un ensemble avec un **ordre** \sqsubseteq , son **diagramme de Hasse** est le graphe ayant \mathbb{D} pour ens. de sommets et dont l'ens des arêtes (orientées de bas en haut) relie tous les A et B vérifiant “ B minimal tq $A \sqsubset B$ ”

Ordre partiel d'un ensemble de parties

Def. Soit \mathbb{D} un ensemble avec un **ordre** \sqsubseteq , son **diagramme de Hasse** est le graphe ayant \mathbb{D} pour ens. de sommets et dont l'ens des arêtes (orientées de bas en haut) relie tous les A et B vérifiant "B minimal tq $A \sqsubset B$ "

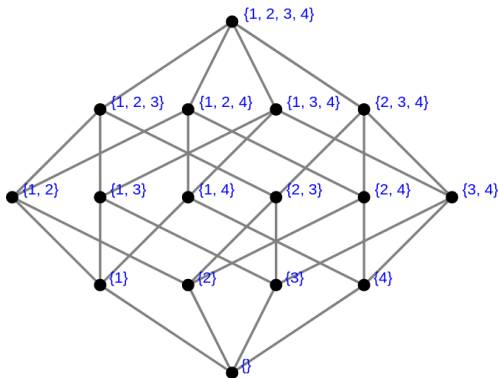


Diagramme de Hasse de $\mathcal{P}(\{1, 2, 3, 4\})$ pour \subseteq

<https://demonstrations.wolfram.com/HasseDiagramOfPowerSets/>

Sa hauteur (= taille maximale d'une suite strictement croissante - 1) vaut 4.

Domaines abstraits comme sous-ordres de $\mathcal{P}(\mathbb{M})$

Exemple pour propriétés de la forme

“ $\bigwedge_i x_i \bowtie_i 0$ ” avec $\bowtie_i \in \{<, >, =, \neq\}$ et $x_i \in \mathbb{Z}$

Domaines abstraits comme sous-ordres de $\mathcal{P}(\mathbb{M})$

Exemple pour propriétés de la forme

$$“\bigwedge_i x_i \bowtie_i 0” \quad \text{avec} \quad \bowtie_i \in \{<, >, =, \neq\} \quad \text{et} \quad x_i \in \mathbb{Z}$$

1) On définit “sous-ordre” de $\mathcal{P}(\mathbb{Z})$ sur

$$\mathbb{V}_0 \stackrel{\text{def}}{=} \{\emptyset, \mathbb{Z}_{<}, \mathbb{Z}_{>}, \mathbb{Z}_{=}, \mathbb{Z}_{\neq}, \mathbb{Z}\},$$

$$\text{où chaque } \mathbb{Z}_{\bowtie} \stackrel{\text{def}}{=} \{v \in \mathbb{Z} \mid v \bowtie 0\}.$$

Domaines abstraits comme sous-ordres de $\mathcal{P}(\mathbb{M})$

Exemple pour propriétés de la forme

$$“\bigwedge_i x_i \bowtie_i 0” \quad \text{avec} \quad \bowtie_i \in \{<, >, =, \neq\} \quad \text{et} \quad x_i \in \mathbb{Z}$$

1) On définit “sous-ordre” de $\mathcal{P}(\mathbb{Z})$ sur

$$\mathbb{V}_0 \stackrel{\text{def}}{=} \{\emptyset, \mathbb{Z}_{<}, \mathbb{Z}_{>}, \mathbb{Z}_{=}, \mathbb{Z}_{\neq}, \mathbb{Z}\},$$

$$\text{où chaque } \mathbb{Z}_{\bowtie} \stackrel{\text{def}}{=} \{v \in \mathbb{Z} \mid v \bowtie 0\}.$$

En fait “sous-ordre” de $\mathcal{P}(\{\mathbb{Z}^{<}, \mathbb{Z}^{>}, \mathbb{Z}^{=}\})$

avec $\mathbb{Z}_{\neq} \equiv (\mathbb{Z}_{<} \cup \mathbb{Z}_{>})$ et $\mathbb{Z} \equiv (\mathbb{Z}_{\neq} \cup \mathbb{Z}_{=})$.

Domaines abstraits comme sous-ordres de $\mathcal{P}(\mathbb{M})$

Exemple pour propriétés de la forme

" $\bigwedge_i x_i \bowtie_i 0$ " avec $\bowtie_i \in \{<, >, =, \neq\}$ et $x_i \in \mathbb{Z}$

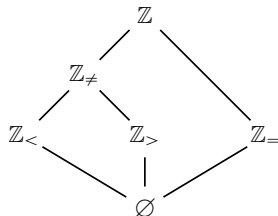
1) On définit "sous-ordre" de $\mathcal{P}(\mathbb{Z})$ sur

$\mathbb{V}_0 \stackrel{\text{def}}{=} \{\emptyset, \mathbb{Z}_{<}, \mathbb{Z}_{>}, \mathbb{Z}_{=}, \mathbb{Z}_{\neq}, \mathbb{Z}\},$

où chaque $\mathbb{Z}_{\bowtie} \stackrel{\text{def}}{=} \{v \in \mathbb{Z} \mid v \bowtie 0\}.$

En fait "sous-ordre" de $\mathcal{P}(\{\mathbb{Z}^{<}, \mathbb{Z}^{>}, \mathbb{Z}^{=}\})$

avec $\mathbb{Z}_{\neq} \equiv (\mathbb{Z}_{<} \cup \mathbb{Z}_{>})$ et $\mathbb{Z} \equiv (\mathbb{Z}_{\neq} \cup \mathbb{Z}_{=}).$



hauteur 3

2) Classiquement, le dom. de *valeurs abstraites* \mathbb{V}_0 s'étend en un dom. d'*états abstraits*, avec

$\mathbb{D}_0 \stackrel{\text{def}}{=} \mathcal{P}(\{m \in \mathbb{M} \mid \bigwedge_i m(x_i) \in V_i\})$ où chaque $V_i \in \mathbb{V}_0.$

un tel dom. est **non-relationnel** car pas de relation entre variables.

Définition des treillis bornés (*bounded lattice*)

Définitions Soit \mathbb{D} est un ensemble muni d'un ordre \sqsubseteq .

- Pour $\mathcal{X} \in \mathcal{P}(\mathbb{D})$ et $B \in \mathbb{D}$, " **B majorant de \mathcal{X}** " ssi $\forall A \in \mathcal{X}, A \sqsubseteq B$.
et " **B minorant de \mathcal{X}** " ssi $\forall A \in \mathcal{X}, B \sqsubseteq A$.

Définition des treillis bornés (*bounded lattice*)

Définitions Soit \mathbb{D} est un ensemble muni d'un ordre \sqsubseteq .

- Pour $\mathcal{X} \in \mathcal{P}(\mathbb{D})$ et $B \in \mathbb{D}$, “**B majorant de \mathcal{X}** ” ssi $\forall A \in \mathcal{X}, A \sqsubseteq B$.
et “**B minorant de \mathcal{X}** ” ssi $\forall A \in \mathcal{X}, B \sqsubseteq A$.
- $(\mathbb{D}, \sqsubseteq)$ est un “**treillis borné**” ssi il admet un minimum noté \perp et un maximum noté \top et si pour tout A_1 et A_2 de \mathbb{D} , alors l'ensemble $\mathcal{X} = \{A_1, A_2\}$ admet un **plus petit majorant** noté $A_1 \sqcup A_2$ et un **plus grand minorant** noté $A_1 \sqcap A_2$.

Définition des treillis bornés (*bounded lattice*)

Définitions Soit \mathbb{D} est un ensemble muni d'un ordre \sqsubseteq .

- Pour $\mathcal{X} \in \mathcal{P}(\mathbb{D})$ et $B \in \mathbb{D}$, “ **B majorant de \mathcal{X}** ” ssi $\forall A \in \mathcal{X}, A \sqsubseteq B$.
et “ **B minorant de \mathcal{X}** ” ssi $\forall A \in \mathcal{X}, B \sqsubseteq A$.
- $(\mathbb{D}, \sqsubseteq)$ est un “**treillis borné**” ssi il admet un minimum noté \perp et un maximum noté \top et si pour tout A_1 et A_2 de \mathbb{D} , alors l'ensemble $\mathcal{X} = \{A_1, A_2\}$ admet un **plus petit majorant** noté $A_1 \sqcup A_2$ et un **plus grand minorant** noté $A_1 \sqcap A_2$.
Autrement dit, $A_1 \sqcup A_2$ (resp. $A_1 \sqcap A_2$) majorant (resp. minorant) de \mathcal{X} et pour tout B majorant (resp. minorant) de \mathcal{X} , alors $A_1 \sqcup A_2 \sqsubseteq B$ (resp. $B \sqsubseteq A_1 \sqcup A_2$).
- $A_1 \sqcup A_2$ (resp. $A_1 \sqcap A_2$) s'appelle aussi leur **borne supérieure (resp. inférieure)**, ou “*least upper (resp. greatest lower) bound*” en anglais. Elles sont **uniques** par définition.

Exemples de treillis bornés

- Pour tout ensemble V , $(\mathcal{P}(V), \subseteq)$ est un treillis borné avec $\perp = \emptyset$, $\top = V$, $\sqcup = \cup$ et $\sqcap = \cap$.

Exemples de treillis bornés

- Pour tout ensemble V , $(\mathcal{P}(V), \subseteq)$ est un treillis borné avec $\perp = \emptyset$, $\top = V$, $\sqcup = \cup$ et $\sqcap = \cap$.
- $(\mathbb{V}_0, \subseteq)$ et $(\mathbb{D}_0, \subseteq)$ de la diapo 14 sont des treillis bornés.

Exemples de treillis bornés

- Pour tout ensemble V , $(\mathcal{P}(V), \subseteq)$ est un treillis borné avec $\perp = \emptyset$, $\top = V$, $\sqcup = \cup$ et $\sqcap = \cap$.
- $(\mathbb{V}_0, \subseteq)$ et $(\mathbb{D}_0, \subseteq)$ de la diapo 14 sont des treillis bornés.
- $(\mathbb{N} \cup \{+\infty\}, \leq)$ est un treillis borné avec $\perp = 0$, $\top = +\infty$, $\sqcup = \max$ et $\sqcap = \min$.

Exemples de treillis bornés

- Pour tout ensemble V , $(\mathcal{P}(V), \subseteq)$ est un treillis borné avec $\perp = \emptyset$, $\top = V$, $\sqcup = \cup$ et $\sqcap = \cap$.
- $(\mathbb{V}_0, \subseteq)$ et $(\mathbb{D}_0, \subseteq)$ de la diapo 14 sont des treillis bornés.
- $(\mathbb{N} \cup \{+\infty\}, \leq)$ est un treillis borné avec $\perp = 0$, $\top = +\infty$, $\sqcup = \max$ et $\sqcap = \min$.
- Classes (hyper-rectangles ou polytopes) de sous-ensembles numériques *convexes* de la diapo 5 sont des treillis bornés de hauteur infinie.

Convergence des suites croissantes dans treillis bornés

pour la convergence de l'inférence d'invariant

Propriété des treillis bornés de hauteur finie

Soit $(\mathbb{D}, \sqsubseteq)$ un treillis borné de hauteur finie h .

Toute suite $(A_n)_{n \in \mathbb{N}}$ croissante sur \mathbb{D} a

au plus $h + 1$ éléments distincts, et donc un maximum !

Convergence des suites croissantes dans treillis bornés

pour la convergence de l'inférence d'invariant

Propriété des treillis bornés de hauteur finie

Soit $(\mathbb{D}, \sqsubseteq)$ un treillis borné de hauteur finie h .

Toute suite $(A_n)_{n \in \mathbb{N}}$ croissante sur \mathbb{D} a

au plus $h + 1$ éléments distincts, et donc un maximum !

Existence de point-fixe (=limite) **non garantie** sur treillis bornés de hauteur infinie !

Contre-exemple : une suite infinie strictement croissante de polyèdres dont l'enveloppe convexe est un cercle !

Convergence des suites croissantes dans treillis bornés

pour la convergence de l'inférence d'invariant

Propriété des treillis bornés de hauteur finie

Soit $(\mathbb{D}, \sqsubseteq)$ un treillis borné de hauteur finie h .

Toute suite $(A_n)_{n \in \mathbb{N}}$ croissante sur \mathbb{D} a

au plus $h + 1$ éléments distincts, et donc un maximum !

Existence de point-fixe (=limite) **non garantie** sur treillis bornés de hauteur infinie !

Contre-exemple : une suite infinie strictement croissante de polyèdres dont l'enveloppe convexe est un cercle !

La littérature fournit des conditions supplémentaires sur les treillis pour garantir l'existence de tels point-fixes.

E.g. **ordre partiel complet** ou *complete partial order*.

Application au choix de ∇_g

D'après diapo 10, trouver A_k tq $\#\{A_0\} S^* \{A_k\}$ exige de construire une suite A_0, A_1, \dots, A_k tq pour tout $i \in 1 \dots k$,
 $\exists A', A_{i+1} = A_i \nabla_{g_i} A'$ et donc $A_i \sqsubseteq A_{i+1}$

Application au choix de ∇_g

D'après diapo 10, trouver A_k tq $\#\{A_0\} S^* \{A_k\}$ exige de construire une suite A_0, A_1, \dots, A_k tq pour tout $i \in 1 \dots k$,

$$\exists A', A_{i+1} = A_i \nabla_{g_i} A' \text{ et donc } A_i \sqsubseteq A_{i+1}$$

Si hauteur finie h , avec $A_1 \nabla_g A_2 \stackrel{def}{=} A_1 \sqcup A_2$,
invariant de boucle trouvé en moins de h itérations !

Application au choix de ∇_g

D'après diapo 10, trouver A_k tq $\#\{A_0\} S^* \{A_k\}$ exige de construire une suite A_0, A_1, \dots, A_k tq pour tout $i \in 1 \dots k$,

$$\exists A', A_{i+1} = A_i \nabla_{g_i} A' \text{ et donc } A_i \sqsubseteq A_{i+1}$$

Si hauteur finie h , avec $A_1 \nabla_g A_2 \stackrel{def}{=} A_1 \sqcup A_2$,
invariant de boucle trouvé en moins de h itérations !

(si on propage les \perp en tête de boucle sans itérer)

Application au choix de ∇_g

D'après diapo 10, trouver A_k tq $\#\{A_0\} S^* \{A_k\}$ exige de construire une suite A_0, A_1, \dots, A_k tq pour tout $i \in 1 \dots k$,
 $\exists A', A_{i+1} = A_i \nabla_{g_i} A'$ et donc $A_i \sqsubseteq A_{i+1}$

Si hauteur finie h , avec $A_1 \nabla_g A_2 \stackrel{def}{=} A_1 \sqcup A_2$,
 invariant de boucle trouvé en moins de h itérations!
 (si on propage les \perp en tête de boucle sans itérer)

Exo Sur diapo 14, hauteur de \mathbb{V}_0 est 3.
 Hauteur de \mathbb{D}_0 pour n variables ?

Application au choix de ∇_g

D'après diapo 10, trouver A_k tq $\#\{A_0\} S^* \{A_k\}$ exige de construire une suite A_0, A_1, \dots, A_k tq pour tout $i \in 1 \dots k$,
 $\exists A', A_{i+1} = A_i \nabla_{g_i} A'$ et donc $A_i \sqsubseteq A_{i+1}$

Si hauteur finie h , avec $A_1 \nabla_g A_2 \stackrel{def}{=} A_1 \sqcup A_2$,
 invariant de boucle trouvé en moins de h itérations!
 (si on propage les \perp en tête de boucle sans itérer)

Exo Sur diapo 14, hauteur de \mathbb{V}_0 est 3.
 Hauteur de \mathbb{D}_0 pour n variables ? $3n$

Application au choix de ∇_g

D'après diapo 10, trouver A_k tq $\#\{A_0\} S^* \{A_k\}$ exige de construire une suite A_0, A_1, \dots, A_k tq pour tout $i \in 1 \dots k$,

$$\exists A', A_{i+1} = A_i \nabla_{g_i} A' \text{ et donc } A_i \sqsubseteq A_{i+1}$$

Si hauteur finie h , avec $A_1 \nabla_g A_2 \stackrel{def}{=} A_1 \sqcup A_2$,
invariant de boucle trouvé en moins de h itérations !

(si on propage les \perp en tête de boucle sans itérer)

Exo Sur diapo 14, hauteur de \mathbb{V}_0 est 3.

Hauteur de \mathbb{D}_0 pour n variables ? **3n**

Dans pire cas, $A_1 \nabla_g A_2$ doit sur-approximer $A_1 \sqcup A_2$
e.g. en conservant contraintes identiques entre A_1 et A_2
mais en supprimant d'autres contraintes. Choix guidable par "g" !

Exemple du calcul de $A[x := T]$ dans \mathbb{D}_0 (défini diapo 14)

Rappel si $A \in \mathbb{D}_0$ avec $A(x) = \mathbb{Z}_{\bowtie}$ alors $\gamma(A(x)) \equiv x \bowtie 0$

On prend $A[x := T] \stackrel{def}{=} \begin{cases} \perp & \text{si } A \sqsubseteq \perp \\ A \oplus \{x \mapsto [T](A)\} & \text{sinon} \end{cases}$

où $[T](A)$ est une sur-approximation de T dans \mathbb{V}_0

Exemple du calcul de $A[x := T]$ dans \mathbb{D}_0 (défini diapo 14)

Rappel si $A \in \mathbb{D}_0$ avec $A(x) = \mathbb{Z}_{\bowtie}$ alors $\gamma(A(x)) \equiv x \bowtie 0$

On prend $A[x := T] \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } A \sqsubseteq \perp \\ A \oplus \{x \mapsto [T](A)\} & \text{sinon} \end{cases}$

où $[T](A)$ est une sur-approximation de T dans \mathbb{V}_0 définie par

$$[x](A) \stackrel{\text{def}}{=} A(x) \quad [-T](A) \stackrel{\text{def}}{=} -[T](A) \quad [T_1 + T_2](A) \stackrel{\text{def}}{=} T_1(A) + T_2(A)$$

$$[n](A) \stackrel{\text{def}}{=} v$$

si $n \in \mathbb{Z}$ et $\exists \bowtie \in \{=, >, <\}$
 tq $v = \mathbb{Z}_{\bowtie} \Leftrightarrow n \bowtie 0$

v	-v
$-\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$
$-\mathbb{Z}_{>}$	$\mathbb{Z}_{<}$
$-\mathbb{Z}_{=}$	$\mathbb{Z}_{=}$
$-\mathbb{Z}_{\neq}$	\mathbb{Z}_{\neq}
$-\top$	\top

$v_1 + v_2$	$\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$	$\mathbb{Z}_{=}$	\mathbb{Z}_{\neq}	\top
$\mathbb{Z}_{<}$	$\mathbb{Z}_{<}$	\top	$\mathbb{Z}_{<}$	\top	\top
$\mathbb{Z}_{>}$	\top	$\mathbb{Z}_{>}$	$\mathbb{Z}_{>}$	\top	\top
$\mathbb{Z}_{=}$	$\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$	$\mathbb{Z}_{=}$	\mathbb{Z}_{\neq}	\top
\mathbb{Z}_{\neq}	\top	\top	\mathbb{Z}_{\neq}	\top	\top
\top	\top	\top	\top	\top	\top

Exemple du calcul de $A[x := T]$ dans \mathbb{D}_0 (défini diapo 14)

Rappel si $A \in \mathbb{D}_0$ avec $A(x) = \mathbb{Z}_{\bowtie}$ alors $\gamma(A(x)) \equiv x \bowtie 0$

On prend $A[x := T] \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } A \sqsubseteq \perp \\ A \oplus \{x \mapsto [T](A)\} & \text{sinon} \end{cases}$

où $[T](A)$ est une sur-approximation de T dans \mathbb{V}_0 définie par

$$[x](A) \stackrel{\text{def}}{=} A(x) \quad [-T](A) \stackrel{\text{def}}{=} -[T](A) \quad [T_1 + T_2](A) \stackrel{\text{def}}{=} T_1(A) + T_2(A)$$

$$[n](A) \stackrel{\text{def}}{=} v$$

si $n \in \mathbb{Z}$ et $\exists \bowtie \in \{=, >, <\}$
 tq $v = \mathbb{Z}_{\bowtie} \Leftrightarrow n \bowtie 0$

v	-v
$-\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$
$-\mathbb{Z}_{>}$	$\mathbb{Z}_{<}$
$-\mathbb{Z}_{=}$	$\mathbb{Z}_{=}$
$-\mathbb{Z}_{\neq}$	\mathbb{Z}_{\neq}
$-\top$	\top

$v_1 + v_2$	$\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$	$\mathbb{Z}_{=}$	\mathbb{Z}_{\neq}	\top
$\mathbb{Z}_{<}$	$\mathbb{Z}_{<}$	\top	$\mathbb{Z}_{<}$	\top	\top
$\mathbb{Z}_{>}$	\top	$\mathbb{Z}_{>}$	$\mathbb{Z}_{>}$	\top	\top
$\mathbb{Z}_{=}$	$\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$	$\mathbb{Z}_{=}$	\mathbb{Z}_{\neq}	\top
\mathbb{Z}_{\neq}	\top	\top	\mathbb{Z}_{\neq}	\top	\top
\top	\top	\top	\top	\top	\top

Rem : calcul de $[T_1 + T_2](A)$ améliorable par
 analyse plus fine de $T_1 + T_2$.

Par exemple, pour $[x + -x](A)$, directement retourner $\mathbb{Z}_{=}$.

Plus généralement, c'est mieux de normaliser les termes affines...

Exemple du calcul de $A[x := T]$ dans \mathbb{D}_0 (défini diapo 14)

Rappel si $A \in \mathbb{D}_0$ avec $A(x) = \mathbb{Z}_{\bowtie}$ alors $\gamma(A(x)) \equiv x \bowtie 0$

On prend $A[x := T] \stackrel{\text{def}}{=} \begin{cases} \perp & \text{si } A \sqsubseteq \perp \\ A \oplus \{x \mapsto [T](A)\} & \text{sinon} \end{cases}$

où $[T](A)$ est une sur-approximation de T dans \mathbb{V}_0 définie par

$$[x](A) \stackrel{\text{def}}{=} A(x) \quad [-T](A) \stackrel{\text{def}}{=} -[T](A) \quad [T_1 + T_2](A) \stackrel{\text{def}}{=} T_1(A) + T_2(A)$$

$$[n](A) \stackrel{\text{def}}{=} v$$

si $n \in \mathbb{Z}$ et $\exists \bowtie \in \{=, >, <\}$
 tq $v = \mathbb{Z}_{\bowtie} \Leftrightarrow n \bowtie 0$

v	-v
$-\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$
$-\mathbb{Z}_{>}$	$\mathbb{Z}_{<}$
$-\mathbb{Z}_{=}$	$\mathbb{Z}_{=}$
$-\mathbb{Z}_{\neq}$	\mathbb{Z}_{\neq}
$-\top$	\top

$v_1 + v_2$	$\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$	$\mathbb{Z}_{=}$	\mathbb{Z}_{\neq}	\top
$\mathbb{Z}_{<}$	$\mathbb{Z}_{<}$	\top	$\mathbb{Z}_{<}$	\top	\top
$\mathbb{Z}_{>}$	\top	$\mathbb{Z}_{>}$	$\mathbb{Z}_{>}$	\top	\top
$\mathbb{Z}_{=}$	$\mathbb{Z}_{<}$	$\mathbb{Z}_{>}$	$\mathbb{Z}_{=}$	\mathbb{Z}_{\neq}	\top
\mathbb{Z}_{\neq}	\top	\top	\mathbb{Z}_{\neq}	\top	\top
\top	\top	\top	\top	\top	\top

Rem : calcul de $[T_1 + T_2](A)$ améliorable par
 analyse plus fine de $T_1 + T_2$.

Par exemple, pour $[x + -x](A)$, directement retourner $\mathbb{Z}_{=}$.

Plus généralement, c'est mieux de normaliser les termes affines...

Quand on ne sait pas traiter T , on pose $[T](A) \stackrel{\text{def}}{=} \top$.

Exemple du calcul de $A \sqcap C$ dans \mathbb{D}_0

Si C est de la forme $T_1 \bowtie T_2$ avec $\bowtie \in \{<, >, =, \neq\}$, alors on fait l'analyse ci-dessous.

Analyse de “ $T_1 \bowtie T_2$ ” on commence par calculer

$v \stackrel{def}{=} [T_1 + -T_2](A) \sqcap \mathbb{Z}_{\bowtie}$. Puis :

1) on retourne \perp si $v = \perp$.

Exemple du calcul de $A \sqcap C$ dans \mathbb{D}_0

Si C est de la forme $T_1 \bowtie T_2$ avec $\bowtie \in \{<, >, =, \neq\}$,
alors on fait l'analyse ci-dessous.

Analyse de " $T_1 \bowtie T_2$ " on commence par calculer

$v \stackrel{def}{=} [T_1 + -T_2](A) \sqcap \mathbb{Z}_{\bowtie}$. Puis :

1) on retourne \perp si $v = \perp$.

2) sinon, si T_1 et T_2 sont deux variables x_1 et x_2 ,
on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\} \sqcap \{x_2 \mapsto [T_1](A) + -v\}$

Exemple du calcul de $A \sqcap C$ dans \mathbb{D}_0

Si C est de la forme $T_1 \bowtie T_2$ avec $\bowtie \in \{<, >, =, \neq\}$, alors on fait l'analyse ci-dessous.

Analyse de " $T_1 \bowtie T_2$ " on commence par calculer

$v \stackrel{def}{=} [T_1 + -T_2](A) \sqcap \mathbb{Z}_{\bowtie}$. Puis :

- 1) on retourne \perp si $v = \perp$.
- 2) sinon, si T_1 et T_2 sont deux variables x_1 et x_2 , on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\} \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 3) sinon, si T_1 est une variable x_1 mais pas T_2 , on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\}$
- 4) sinon, si T_2 est une variable x_2 mais pas T_1 , on retourne $A \sqcap \{x_2 \mapsto [T_1](A) + -v\}$

Exemple du calcul de $A \sqcap C$ dans \mathbb{D}_0

Si C est de la forme $T_1 \bowtie T_2$ avec $\bowtie \in \{<, >, =, \neq\}$, alors on fait l'analyse ci-dessous.

Analyse de " $T_1 \bowtie T_2$ " on commence par calculer

$v \stackrel{\text{def}}{=} [T_1 + -T_2](A) \sqcap \mathbb{Z}_{\bowtie}$. Puis :

- 1) on retourne \perp si $v = \perp$.
- 2) sinon, si T_1 et T_2 sont deux variables x_1 et x_2 , on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\} \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 3) sinon, si T_1 est une variable x_1 mais pas T_2 , on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\}$
- 4) sinon, si T_2 est une variable x_2 mais pas T_1 , on retourne $A \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 5) sinon, on retourne A .

Exemple du calcul de $A \sqcap C$ dans \mathbb{D}_0

Si C est de la forme $T_1 \bowtie T_2$ avec $\bowtie \in \{<, >, =, \neq\}$,
alors on fait l'analyse ci-dessous.

Analyse de " $T_1 \bowtie T_2$ " on commence par calculer

$v \stackrel{\text{def}}{=} [T_1 + -T_2](A) \sqcap \mathbb{Z}_{\bowtie}$. Puis :

- 1) on retourne \perp si $v = \perp$.
- 2) sinon, si T_1 et T_2 sont deux variables x_1 et x_2 ,
on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\} \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 3) sinon, si T_1 est une variable x_1 mais pas T_2 ,
on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\}$
- 4) sinon, si T_2 est une variable x_2 mais pas T_1 ,
on retourne $A \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 5) sinon, on retourne A .

Rem : l'analyse ci-dessus retourne de meilleurs résultats pour " $x_1 \bowtie x_2$ "
que pour " $x_1 + -x_2 \bowtie 0$ " ce qui prouve qu'on peut encore l'améliorer...

Exemple du calcul de $A \sqcap C$ dans \mathbb{D}_0

Si C est de la forme $T_1 \bowtie T_2$ avec $\bowtie \in \{<, >, =, \neq\}$,
 alors on fait l'analyse ci-dessous. Sinon, pour $\neg(T_1 \bowtie' T_2)$,
 si $\exists \bowtie, (\mathbb{Z} \setminus \mathbb{Z}_{\bowtie'}) \subseteq \mathbb{Z}_{\bowtie}$, on se ramène alors à l'analyse de $T_1 \bowtie T_2$.
 Sinon, on retourne A .

Analyse de " $T_1 \bowtie T_2$ " on commence par calculer

$v \stackrel{\text{def}}{=} [T_1 + -T_2](A) \sqcap \mathbb{Z}_{\bowtie}$. Puis :

- 1) on retourne \perp si $v = \perp$.
- 2) sinon, si T_1 et T_2 sont deux variables x_1 et x_2 ,
 on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\} \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 3) sinon, si T_1 est une variable x_1 mais pas T_2 ,
 on retourne $A \sqcap \{x_1 \mapsto v + [T_2](A)\}$
- 4) sinon, si T_2 est une variable x_2 mais pas T_1 ,
 on retourne $A \sqcap \{x_2 \mapsto [T_1](A) + -v\}$
- 5) sinon, on retourne A .

Rem : l'analyse ci-dessus retourne de meilleurs résultats pour " $x_1 \bowtie x_2$ "
 que pour " $x_1 + -x_2 \bowtie 0$ " ce qui prouve qu'on peut encore l'améliorer...

Plan du Cours 3

Idées de base

Interpréteur “en avant” intra-procédural générique et correct

Théorie des *treillis bornés* et applications aux domaines abstraits

Analyses de valeurs et applications

Analyses des variables vivantes et applications

Combiner analyses du “flot de donnée” et du “flot de contrôle”

Esquisse d'une analyse de valeurs et d'alias à la EVA

Cf. <https://frama-c.com/download/frama-c-eva-manual.pdf>

Domaine non-relationnel

avec Valeurs Abstraites (VA) associées,

soit aux variables **scalaires jamais** prises par adresse

soit aux adresses b_x | de variables x (alors recodées en $*b_x$)
ou issues d'un malloc

Esquisse d'une analyse de valeurs et d'alias à la EVA

Cf. <https://frama-c.com/download/frama-c-eva-manual.pdf>

Domaine non-relationnel

avec **Valeurs Abstraites (VA)** associées,

soit aux variables **scalaires jamais** prises par adresse

soit aux adresses b_x | de variables x (alors recodées en $*b_x$)
ou issues d'un malloc

VA d'entier : énumération finies d'entiers possibles ;

ou intervalle d'entiers $[l, u]$ avec $l, u \in \mathbb{Z}$; ...

VA de flottant : intervalle de flottants

VA de pointeur : disjonction finie d'adresses où chaque adresse de la forme $b_x + o$ ou $\text{NULL} + o$ avec o valeur abstraite de l'offset

VA de struct/array : tableau de VA de scalaires associé à un b_x .

Démo avec plugin EVA de Frama-C (25.0)

Analyse de ces fichiers avec “frama-c -eva” ou “frama-c-gui -eva”

- ▶ `pointeurs_eva.c` : UAF (Use-After-Free) détecté (en rouge)!

- ▶ `binary_search_eva_ko0.c` :

```
[eva:alarm] binary_search_eva_ko0.c:34: Warning:
  accessing out of bounds index. assert 0 ≤ i;
...
1 alarm generated by the analysis:
  1 access out of bounds index
```

- ▶ `binary_search_eva_ok.c` :

```
0 alarms generated by the analysis.
```

- ▶ `binary_search_eva_ko1.c` :

```
[eva:alarm] binary_search_eva_ko1.c:9: Warning:
  out of bounds read. assert \valid_read(t + m);
...
1 alarm generated by the analysis:
  1 invalid memory access
1 of them is a sure alarm (invalid status).
```

Applications des analyses statiques de valeur

Sûreté de fonctionnement

Prouver l'absence de RTE sur systèmes de contrôles commandes embarqués (e.g. Airbus A340 et A380), cf. Astrée.

Détection de vulnérabilités logicielles (sécurité)

En combinaison avec d'autres techniques, pour éviter trop de fausses alarmes. cf. Coverity.

Optimisations de compilation (cf. diapos suivantes)

Déplacement de code (LCM), propagation de constantes, élimination de code mort (DCE), ...
dans GCC, CLANG/LLVM, et même (Chamois) CompCert

Lazy Code Motion avec Strength Reduction

Dans un contexte “size_t i, n” et “float *p, *t”,
pour optimiser dans la boucle ci-dessous
accès mémoire “*p” et calculs des adresses “t+i”

Avec code initial

```
while (i != n) {  
    *p += *(t+i);  
    i++;  
}
```

Lazy Code Motion avec Strength Reduction

Dans un contexte “size_t i, n” et “float *p, *t”,
pour optimiser dans la boucle ci-dessous
accès mémoire “*p” et calculs des adresses “t+i”

Avec code initial

```
while (i != n) {  
    *p += *(t+i);  
    i++;  
}
```

LCM après déroulement du 1^o tour de boucle

```
if (i != n){  
    register float r1=*p,*r2=t+i,*r3=t+n;  
    while (r2 != r3) {  
        r1 += *r2;  
        r2++;  
    }  
    *p=r1;  
    i=n;  
}
```

Lazy Code Motion avec Strength Reduction

Dans un contexte “size_t i, n” et “float *p, *t”,
pour optimiser dans la boucle ci-dessous
accès mémoire “*p” et calculs des adresses “t+i”

Avec code initial

```
while (i != n) {
    *p += *(t+i);
    i++;
}
```

LCM après déroulement du 1^o tour de boucle

```
if (i != n){
    register float r1=*p,*r2=t+i,*r3=t+n;
    while (r2 != r3) {
        r1 += *r2;
        r2++;
    }
    *p=r1;
    i=n;
}
```

Correct sous hypothèse que bases des adresses “b...” des VA de
&i, &n, p et t sont **2 à 2 distinctes**

Propagation de constantes & DCE (après déroulements)

Exemple sur compilation de “x=fact(2);”

Dans contexte

après inlining & déroulement du 1^o tour de boucle

```
long fact(long n) {  
    long r=n;  
    while (n>1) r*=-n;  
    return r;  
}
```

```
{  
    long n=2, r=n;  
    if (n>1){r*=-n; while (n>1) r*=-n;}  
    x=r;  
}
```

Propagation de constantes & DCE (après déroulements)

Exemple sur compilation de "x=fact(2);"

Dans contexte

après inlining & déroulement du 1^o tour de boucle

```
long fact(long n) {
  long r=n;
  while (n>1) r*=-n;
  return r;
}
```

```
{
  long n=2, r=n;
  if (n>1){r*=-n; while (n>1) r*=-n;}
  x=r;
}
```

Propagation de constantes

= propagation des VA singletons

```
{ long n=2, r=2;
  if (2>1){r=2;n=1; while (1>1) r*=-n;}
  x=2;}
```

Propagation de constantes & DCE (après déroulements)

Exemple sur compilation de "x=fact(2);"

Dans contexte

après inlining & déroulement du 1^o tour de boucle

```
long fact(long n) {
  long r=n;
  while (n>1) r*=-n;
  return r;
}
```

```
{
  long n=2, r=n;
  if (n>1){r*=-n; while (n>1) r*=-n;}
  x=r;
}
```

Propagation de constantes

= propagation des VA singletons

DCE = élim. gardes vides

```
{ long n=2, r=2;
  if (2>1){r=2;n=1; while (1>1) r*=-n;}
  x=2;}
```

```
{ long n=2, r=2;
  r=2;n=1;
  x=2;}
```

Propagation de constantes & DCE (après déroulements)

Exemple sur compilation de "x=fact(2);"

Dans contexte

après inlining & déroulement du 1^o tour de boucle

```
long fact(long n) {
  long r=n;
  while (n>1)r*=-n;
  return r;
}
```

```
{
  long n=2, r=n;
  if (n>1){r*=-n; while (n>1)r*=-n;}
  x=r;
}
```

Propagation de constantes

= propagation des VA singletons

DCE = élim. gardes vides

```
{ long n=2, r=2;
  if (2>1){r=2;n=1; while (1>1)r*=-n;}
  x=2;}
```

```
{ long n=2, r=2;
  r=2;n=1;
  x=2;}
```

Après analyse de liveness & élim. affectations inutiles (cf. diapo 35)

"x=fact(2);" finalement compilé en "x=2;"

Plan du Cours 3

Idées de base

Interpréteur “en avant” intra-procédural générique et correct

Théorie des *treillis bornés* et applications aux domaines abstraits

Analyses de valeurs et applications

Analyses des variables vivantes et applications

Combiner analyses du “flot de donnée” et du “flot de contrôle”

Exemple d'UB (en C) par variable non-initialisée

```
1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }
```

Exemple d'UB (en C) par variable non-initialisée

```
1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }
```

```
gcc -Wall -Wextra -pedantic -std=c99
```

↪ main retourne "42" sans warning!

Exemple d'UB (en C) par variable non-initialisée

```
1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }
```

```
gcc -Wall -Wextra -pedantic -std=c99
```

↪ main retourne "42" sans warning!

```
gcc -O1 -Wall
```

```
warning: 'k' is used uninitialized line 10
```

↪ nombre différent à chaque appel

Exemple d'UB (en C) par variable non-initialisée

```

1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }

```

```
gcc -Wall -Wextra -pedantic -std=c99
```

↪ main retourne "42" sans warning!

```
gcc -O1 -Wall
```

```
warning: 'k' is used uninitialized line 10
```

↪ nombre différent à chaque appel

```
frama-c -wp -wp-rte
```

```
[wp] Proved goals:    3 / 3
```

Exemple d'UB (en C) par variable non-initialisée

```

1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }

```

```
gcc -Wall -Wextra -pedantic -std=c99
```

↪ main retourne "42" sans warning!

```
gcc -O1 -Wall
```

```
warning: 'k' is used uninitialized line 10
```

↪ nombre différent à chaque appel

```
frama-c -wp -wp-rte
```

```
[wp] Proved goals:    3 / 3
```

↪ UB non vérifié par plugin RTE!

Exemple d'UB (en C) par variable non-initialisée

```

1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }

```

```
gcc -Wall -Wextra -pedantic -std=c99
```

↪ main retourne "42" sans warning!

```
gcc -O1 -Wall
```

```
warning: 'k' is used uninitialized line 10
```

↪ nombre différent à chaque appel

```
frama-c -wp -wp-rte
```

```
[wp] Proved goals: 3 / 3
```

↪ UB non vérifié par plugin RTE!

```
frama-c -eva
```

```
accessing uninitialized left-value line 10.
1 sure alarm (invalid status).
```

Exemple d'UB (en C) par variable non-initialisée

```

1  bool ok(long pwd){
2      long k=42; //CLÉ SECRÈTE
3      return pwd==k;
4  }
5
6  /*@assigns \nothing;*/
7  unsigned int foo(int c) {
8      unsigned int k;
9      /*@loop assigns i,k;*/
10     for(int i=0;i<c;i++)k*=c;
11     return k;
12 }
13
14 int main(){
15     if (ok(25)) return 0;
16     return foo(1);
17 }

```

```
gcc -Wall -Wextra -pedantic -std=c99
```

↪ main retourne "42" sans warning!

```
gcc -O1 -Wall
```

```
warning: 'k' is used uninitialized line 10
```

↪ nombre différent à chaque appel

```
frama-c -wp -wp-rte
```

```
[wp] Proved goals: 3 / 3
```

↪ UB non vérifié par plugin RTE!

```
frama-c -eva
```

```
accessing uninitialized left-value line 10.
1 sure alarm (invalid status).
```

Très dangereux pour la sécurité!

fuites d'informations

+ **attaques** sur pointeurs non-initialisés de type "UAF" (cf. TP)

Exemple de variable partiellement initialisée sans/avec UB

(comme trouvé par "frama-c -eva")

```
1 union {  
2     short n;  
3     double f;  
4 } x, y;  
5 x.n = 42;  
6 y = x;  
7 y.n /= 6;  
8 return y.n;
```

Ci-contre : **sans UB** (qui retourne 7)

Exemple de variable partiellement initialisée sans/avec UB

(comme trouvé par “frama-c -eva”)

```
1  union {
2    short n;
3    double f;
4  } x, y;
5  x.n = 42;
6  y = x;
7  y.n /= 6;
8  return y.n;
```

Ci-contre : **sans UB** (qui retourne 7)

mais pas si ligne 7 remplacée par “y.f /= 6;”

Exemple de variable partiellement initialisée sans/avec UB

(comme trouvé par “frama-c -eva”)

```
1 union {
2     short n;
3     double f;
4 } x, y;
5 x.n = 42;
6 y = x;
7 y.n /= 6;
8 return y.n;
```

Ci-contre : **sans UB** (qui retourne 7)

mais pas si ligne 7 remplacée par “y.f /= 6;”

“frama-c -eva” signale bien cet UB

mais pas “gcc -O3 -Wall -Wextra”

Exemple de variable partiellement initialisée sans/avec UB

(comme trouvé par "frama-c -eva")

```
1 union {  
2   short n;  
3   double f;  
4 } x, y;  
5 x.n = 42;  
6 y = x;  
7 y.n /= 6;  
8 return y.n;
```

Ci-contre : **sans UB** (qui retourne 7)

mais pas si ligne 7 remplacée par "y.f /= 6;"

"frama-c -eva" signale bien cet UB

mais pas "gcc -O3 -Wall -Wextra"

Et en "-O0" exécutable retourne

7 ou 92 ou 178 ou ...

Exemple de variable partiellement initialisée sans/avec UB

(comme trouvé par “frama-c -eva”)

```

1  union {
2     short  n;
3     double f;
4  } x, y;
5  x.n = 42;
6  y = x;
7  y.n /= 6;
8  return y.n;

```

Ci-contre : **sans UB** (qui retourne 7)

mais pas si ligne 7 remplacée par “y.f /= 6;”

“frama-c -eva” signale bien cet UB

mais pas “gcc -O3 -Wall -Wextra”

Et en “-O0” exécutable retourne

7 ou 92 ou 178 ou ...

A bas niveau, nécessité

de manipuler données **partiellement initialisées !**

(pour des questions d'alignement notamment)

Exemple de variable partiellement initialisée sans/avec UB

(comme trouvé par "frama-c -eva")

```

1  union {
2     short  n;
3     double f;
4  } x, y;
5  x.n = 42;
6  y = x;
7  y.n /= 6;
8  return y.n;

```

Ci-contre : **sans UB** (qui retourne 7)

mais pas si ligne 7 remplacée par "y.f /= 6;"

"frama-c -eva" signale bien cet UB

mais pas "gcc -O3 -Wall -Wextra"

Et en "-O0" exécutable retourne

7 ou 92 ou 178 ou ...

A bas niveau, nécessité

de manipuler données **partiellement initialisées !**

(pour des questions d'alignement notamment)

Motive la norme C qui indique **sans UB** si

non-initialisation est "*non observable*"

e.g. non-initialisation interdite dans :

return d'une fonction, paramètre de fonction externe,
condition de branchement, déref de pointeur, etc.

Formalisation par valeur \perp et commande **obs**

Sémantique introduit une valeur “**indéfini**” spéciale, notée ici \perp , seulement utilisée pour initialiser les variables à valeur par défaut, et **absorbante** pour tout opérateur, e.g.

$$0 \times \perp = \perp$$

$$\perp - x = \perp$$

Formalisation par valeur \perp et commande **obs**

Sémantique introduit une valeur “**indéfini**” spéciale, notée ici \perp , seulement utilisée pour initialiser les variables à valeur par défaut, et **absorbante** pour tout opérateur, e.g.

$$0 \times \perp = \perp$$

$$\perp - x = \perp$$

Diapo suivante : variante de triplets de Hoare, notés $\langle P \rangle S \langle Q \rangle$, pour garantir \perp non-observable (sous précondition P) :

Formalisation par valeur \perp et commande **obs**

Sémantique introduit une valeur “**indéfini**” spéciale, notée ici \perp , seulement utilisée pour initialiser les variables à valeur par défaut, et **absorbante** pour tout opérateur, e.g.

$$0 \times \perp = \perp$$

$$\perp - x = \perp$$

Diapo suivante : variante de triplets de Hoare, notés $\langle P \rangle S \langle Q \rangle$, pour garantir \perp non-observable (sous précondition P) :

- ▶ nouvelle commande gardée “**obs** T ” qui déclare T observable avec “**assert**($T \neq \perp$)” (le non-observable étant par défaut).
mais équivalente à ε pour triplets $\{P\} S \{Q\}$ **usuels**.

Formalisation par valeur \perp et commande **obs**

Sémantique introduit une valeur “**indéfini**” spéciale, notée ici \perp , seulement utilisée pour initialiser les variables à valeur par défaut, et **absorbante** pour tout opérateur, e.g.

$$0 \times \perp = \perp$$

$$\perp - x = \perp$$

Diapo suivante : variante de triplets de Hoare, notés $\langle P \rangle S \langle Q \rangle$, pour garantir \perp non-observable (sous précondition P) :

- ▶ nouvelle commande gardée “**obs** T ” qui déclare T observable avec “**assert**($T \neq \perp$)” (le non-observable étant par défaut).
mais équivalente à ε pour triplets $\{P\} S \{Q\}$ **usuels**.
- ▶ pour L ens. de variables, on note **alive**(L) $\stackrel{def}{=} \bigwedge_{x \in L} x \neq \perp$
- ▶ on note $\mathcal{V}(T)$ et $\mathcal{V}(C)$ ens. des variables (libres) de T et C .

Logique de Hoare avec garantie \Downarrow non-observable

Règles spécifiques

$$\frac{}{\langle \mathbf{alive}(\mathcal{V}(T)) \wedge Q \rangle \mathbf{obs} T \langle Q \rangle}$$

$$\frac{}{\langle \mathbf{alive}(\mathcal{V}(C)) \wedge (C \Rightarrow Q) \rangle \mathbf{assume} C \langle Q \rangle}$$

Logique de Hoare avec garantie $\not\vdash$ non-observable

Règles spécifiques

$$\overline{\langle \mathbf{alive}(\mathcal{V}(T)) \wedge Q \rangle \mathbf{obs} T \langle Q \rangle}$$

$$\overline{\langle \mathbf{alive}(\mathcal{V}(C)) \wedge (C \Rightarrow Q) \rangle \mathbf{assume} C \langle Q \rangle}$$

Règles inchangées

$$\frac{\langle P' \rangle S \langle Q' \rangle}{\langle P \rangle S \langle Q \rangle} \quad \begin{array}{l} P \Rightarrow P' \\ Q' \Rightarrow Q \end{array}$$

$$\overline{\langle \mathbf{false} \rangle \mathbf{abort} \langle Q \rangle}$$

$$\overline{\langle Q[x \leftarrow T] \rangle x := T \langle Q \rangle}$$

$$\frac{\langle P \rangle S_1 \langle I \rangle \quad \langle I \rangle S_2 \langle Q \rangle}{\langle P \rangle S_1; S_2 \langle Q \rangle}$$

$$\frac{\langle P \rangle S_1 \langle Q \rangle \quad \langle P \rangle S_2 \langle Q \rangle}{\langle P \rangle S_1 \sqcup S_2 \langle Q \rangle}$$

$$\frac{\langle I \rangle S \langle I \rangle}{\langle I \rangle S^* \langle I \rangle}$$

Analyse (de base) des variables vivantes (*liveness*)

Pour L_i et L_o ensemble de variables, on définit diapo suivante un système d'inférence noté $\heartsuit\langle L_i \rangle S \langle L_o \rangle$ tel que

$$\heartsuit\langle L_i \rangle S \langle L_o \rangle \wedge \{P\} S \{Q\} \Rightarrow \langle \mathbf{alive}(L_i) \wedge P \rangle S \langle \mathbf{alive}(L_o) \wedge Q \rangle$$

Analyse (de base) des variables vivantes (*liveness*)

Pour L_i et L_o ensemble de variables, on définit diapo suivante un système d'inférence noté $\heartsuit\langle L_i \rangle S \langle L_o \rangle$ tel que

$$\heartsuit\langle L_i \rangle S \langle L_o \rangle \wedge \{P\} S \{Q\} \Rightarrow \langle \mathbf{alive}(L_i) \wedge P \rangle S \langle \mathbf{alive}(L_o) \wedge Q \rangle$$

Sur triplet $\heartsuit\langle L_i \rangle S \langle L_o \rangle$

- ▶ L_i ensemble des variables vivantes en entrée (*live-in*)
- ▶ L_o ensemble des variables vivantes en sortie (*live-out*)

Analyse (de base) des variables vivantes (*liveness*)

Pour L_i et L_o ensemble de variables, on définit diapo suivante un système d'inférence noté $\heartsuit\langle L_i \rangle S \langle L_o \rangle$ tel que

$$\heartsuit\langle L_i \rangle S \langle L_o \rangle \wedge \{P\} S \{Q\} \Rightarrow \langle \mathbf{alive}(L_i) \wedge P \rangle S \langle \mathbf{alive}(L_o) \wedge Q \rangle$$

Sur triplet $\heartsuit\langle L_i \rangle S \langle L_o \rangle$

- ▶ L_i ensemble des variables vivantes en entrée (*live-in*)
- ▶ L_o ensemble des variables vivantes en sortie (*live-out*)

Interprétation abstraite **en arrière** dans treillis borné $\mathcal{P}(\mathbb{X})$,
où \mathbb{X} ensemble (fini) des variables du programme
i.e. treillis de hauteur $\text{card}(\mathbb{X})$.

Analyse (de base) des variables vivantes (*liveness*)

Pour L_i et L_o ensemble de variables, on définit diapo suivante un système d'inférence noté $\heartsuit\langle L_i \rangle S \langle L_o \rangle$ tel que

$$\heartsuit\langle L_i \rangle S \langle L_o \rangle \wedge \{P\} S \{Q\} \Rightarrow \langle \mathbf{alive}(L_i) \wedge P \rangle S \langle \mathbf{alive}(L_o) \wedge Q \rangle$$

Sur triplet $\heartsuit\langle L_i \rangle S \langle L_o \rangle$

- ▶ L_i ensemble des variables vivantes en entrée (*live-in*)
- ▶ L_o ensemble des variables vivantes en sortie (*live-out*)

Interprétation abstraite **en arrière** dans treillis borné $\mathcal{P}(\mathbb{X})$,
où \mathbb{X} ensemble (fini) des variables du programme
i.e. treillis de hauteur $\text{card}(\mathbb{X})$.

$$\text{On note } L_1[x \leftarrow L_2] \stackrel{\text{def}}{=} \begin{cases} L_1 & \text{si } x \notin L_1 \\ (L_1 \setminus \{x\}) \cup L_2 & \text{sinon} \end{cases}$$

Règles en arrière (de base) pour l'inférence de *liveness*

$$\overline{\heartsuit \langle \mathcal{V}(C) \cup L \rangle \text{ assume } C \langle L \rangle}$$

$$\overline{\heartsuit \langle \mathcal{V}(T) \cup L \rangle \text{ obs } T \langle L \rangle}$$

$$\overline{\heartsuit \langle \emptyset \rangle \text{ abort } \langle L \rangle}$$

$$\overline{\heartsuit \langle L[x \leftarrow \mathcal{V}(T)] \rangle x := T \langle L \rangle}$$

$$\frac{\heartsuit \langle L_0 \rangle S_1 \langle L_1 \rangle \quad \heartsuit \langle L_1 \rangle S_2 \langle L_2 \rangle}{\heartsuit \langle L_0 \rangle S_1; S_2 \langle L_2 \rangle}$$

$$\frac{\heartsuit \langle L_1 \rangle S_1 \langle L \rangle \quad \heartsuit \langle L_2 \rangle S_2 \langle L \rangle}{\heartsuit \langle L_1 \cup L_2 \rangle S_1 \sqcup S_2 \langle L \rangle}$$

$$\frac{\heartsuit \langle L_i \rangle S \langle L \rangle}{\heartsuit \langle L \rangle S^* \langle L \rangle} \quad L_i \subseteq L$$

$$\frac{\heartsuit \langle L_i \rangle S^* \langle L_0 \cup L \rangle \quad \heartsuit \langle L_0 \rangle S \langle L \rangle}{\heartsuit \langle L_i \rangle S^* \langle L \rangle}$$

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

```
1 while (f1 <= n){
2     x = f0+f1;
3     f0 = f1;
4     f1 = x;
5     i++;
6 }
7 // f1 > n
8 printf("%u",f0);
```

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

```
1 while (f1 <= n){
2     x = f0+f1;
3     f0 = f1;
4     f1 = x;
5     i++;
6 }
7 // f1 > n
8 printf("%u",f0);
```

∅

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

```
1  while (f1 <= n){
2      x = f0+f1;
3      f0 = f1;
4      f1 = x;
5      i++;
6  }
7  // f1 > n           {f0}
8  printf("%u",f0);   ∅
```

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f_0).

```
1 while (f1 <= n){
2     x = f0+f1;
3     f0 = f1;
4     f1 = x;
5     i++;
6 }
7 // f1 > n
8 printf("%u", f0);
```

{ f_0, f_1, n }

{ f_0 }

\emptyset

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f_0).

1 while (f1 <= n){ 2 x = f0+f1; 3 f0 = f1; 4 f1 = x; 5 i++; 6 } 7 // f1 > n 8 printf(“%u”,f0);	{f0, f1, n} {f0, f1, n} {f0} ∅
--	---

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

<pre> 1 while (f1 <= n){ 2 x = f0+f1; 3 f0 = f1; 4 f1 = x; 5 i++; 6 } 7 // f1 > n 8 printf("%u",f0); </pre>	<pre> {f0, f1, n} {f0, f1, n} {f0, f1, n} {f0} ∅ </pre>
---	---

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

<pre> 1 while (f1 <= n){ 2 x = f0+f1; 3 f0 = f1; 4 f1 = x; 5 i++; 6 } 7 // f1 > n 8 printf("%u",f0); </pre>	<pre> {f0,x,n} {f0,f1,n} {f0,f1,n} {f0,f1,n} {f0} ∅ </pre>
---	--

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

<pre> 1 while (f1 <= n){ 2 x = f0+f1; 3 f0 = f1; 4 f1 = x; 5 i++; 6 } 7 // f1 > n 8 printf("%u",f0); </pre>	<pre> {f1,x,n} {f0,x,n} {f0,f1,n} {f0,f1,n} {f0,f1,n} {f0,f1,n} {f0} ∅ </pre>
---	---

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

1	while (f1 <= n){	{f0, f1, n}
2	x = f0+f1;	{f1, x, n}
3	f0 = f1;	{f0, x, n}
4	f1 = x;	{f0, f1, n}
5	i++;	{f0, f1, n}
6	}	{f0, f1, n}
7	// f1 > n	{f0}
8	printf(“%u”, f0);	∅

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

1	while (f1 <= n){	{f0, f1, n}
2	x = f0+f1;	{f1, x, n}
3	f0 = f1;	{f0, x, n}
4	f1 = x;	{f0, f1, n}
5	i++;	{f0, f1, n}
6	}	{f0, f1, n}
7	// f1 > n	{f0}
8	printf(“%u”, f0);	∅

Invariant à l'entrée : {f0, f1, n}.

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

<pre> 1 while (f1 <= n){ 2 x = f0+f1; 3 f0 = f1; 4 f1 = x; 5 i++; 6 } 7 // f1 > n 8 printf("%u", f0); </pre>	<pre> {f0, f1, n} {f1, x, n} {f0, x, n} {f0, f1, n} {f0, f1, n} {f0, f1, n} {f0, f1, n} {f0} ∅ </pre>
--	---

Invariant à l'entrée : $\{f0, f1, n\}$.

Exo : avec **obs**(i) au lieu de **obs**(f0) en ligne 8, on aurait ...

Exemple du calcul des variables *live-in*

sur le calcul du plus grand nombre de Fibonacci majoré par un entier n

Toutes var en “unsigned int” et seule ligne 8 fait un **obs**(f0).

1	while (f1 <= n){	{f0, f1, n}
2	x = f0+f1;	{f1, x, n}
3	f0 = f1;	{f0, x, n}
4	f1 = x;	{f0, f1, n}
5	i++;	{f0, f1, n}
6	}	{f0, f1, n}
7	// f1 > n	{f0}
8	printf(“%u”, f0);	∅

Invariant à l'entrée : {f0, f1, n}.

Exo : avec **obs**(i) au lieu de **obs**(f0) en ligne 8, on aurait ...

...{i, f0, f1, n} en 2 tours de boucle

Applications des analyses de variables vivantes

Applications en compilation

- ▶ Élimination des affectations de **registres** inutiles :
A l'issue de l'analyse,
remplacer par " ε " tout " $x := T$ " avec live-out sans " x ".

Applications des analyses de variables vivantes

Applications en compilation

- ▶ Élimination des affectations de **registres** inutiles :
A l'issue de l'analyse,
remplacer par " ε " tout " $x := T$ " avec live-out sans " x ".

Sur diapo précédente, pour $\{f0\}$ en live-out de la boucle :
i (ligne 6) éliminable, mais pas x (ligne 3).

Applications des analyses de variables vivantes

Applications en compilation

- ▶ Élimination des affectations de **registres** inutiles :
A l'issue de l'analyse,
remplacer par " ε " tout " $x := T$ " avec live-out sans " x ".
- Sur diapo précédente, pour $\{f0\}$ en live-out de la boucle :
- ▶ Généralisable à élimination des **stores inutiles** :
en combinant avec *analyse de valeurs et d'alias*.

Applications des analyses de variables vivantes

Applications en compilation

- ▶ Élimination des affectations de **registres** inutiles :
A l'issue de l'analyse,
remplacer par "ε" tout "x := T" avec live-out sans "x".

Sur diapo précédente, pour $\{f0\}$ en live-out de la boucle :
i (ligne 6) éliminable, mais pas x (ligne 3).
- ▶ Généralisable à élimination des **stores inutiles** :
en combinant avec *analyse de valeurs et d'alias*.

Applications (de variantes) en sécurité : **les analyses de teintes**

- ▶ recherche de *fuites d'information*
par marquage "*données publiques*" avec **obs**
pour vérifier absence de "*données secrètes*" dans *live-in*.

Applications des analyses de variables vivantes

Applications en compilation

- ▶ Élimination des affectations de **registres** inutiles :
A l'issue de l'analyse,
remplacer par "ε" tout "x := T" avec live-out sans "x".

Sur diapo précédente, pour $\{f0\}$ en live-out de la boucle :
i (ligne 6) éliminable, mais pas x (ligne 3).
- ▶ Généralisable à élimination des **stores inutiles** :
en combinant avec *analyse de valeurs et d'alias*.

Applications (de variantes) en sécurité : **les analyses de teintes**

- ▶ recherche de *fuites d'information*
par marquage "*données publiques*" avec **obs**
pour vérifier absence de "*données secrètes*" dans *live-in*.
- ▶ ou de vulnérabilités aux *injections malveillantes*
par marquage "*données critiques*" avec **obs**
à protéger de "*données malveillantes*" dans *live-in*.

Plan du Cours 3

Idées de base

Interpréteur “en avant” intra-procédural générique et correct

Théorie des *treillis bornés* et applications aux domaines abstraits

Analyses de valeurs et applications

Analyses des variables vivantes et applications

Combiner analyses du “flot de donnée” et du “flot de contrôle”

Analyses du “flot de donnée” (*data-flow*)

Récapitulation du cadre général des analyses vues jusqu’ici

Définition informelle

Inférer ensembles “d’informations” en chaque point de contrôle par “approximations” de l’exécution concrète dans un *treillis borné*, avec des calculs de points-fixes (“invariants”).

Analyses du “flot de donnée” (*data-flow*)

Récapitulation du cadre général des analyses vues jusqu’ici

Définition informelle

Inférer ensembles “d’informations” en chaque point de contrôle par “approximations” de l’exécution concrète dans un *treillis borné*, avec des calculs de points-fixes (“invariants”).

Caractéristiques (corrélées entre elles) d’une telle analyse

- ▶ Classe des “ensembles” collectés ?
- ▶ Garanties : sur toute exécution ? au moins une exécution ?
- ▶ Sur ou sous-approximation ?
- ▶ Propagation avant ou arrière ?
- ▶ Interprétation du $S_1 \sqcup S_2$ en union ou intersection ?
- ▶ Initialisation du calcul en \top ou \perp ?

Graphe de flot de contrôle (sur un exemple)

Définition d'un CFG (*Control Flow Graph*)

Représente le **corps** de chaque fonction par un **graphe orienté** où

nœud = **bloc** avec 1 unique point d'entrée sur une séquence d'affectations, éventuellement terminée par une instruction de branchement.

arc = branchement éventuellement pris selon une condition.

Graphe de flot de contrôle (sur un exemple)

Définition d'un CFG (*Control Flow Graph*)

Représente le **corps** de chaque fonction par un **graphe orienté** où

nœud = **bloc** avec 1 unique point d'entrée sur une séquence d'affectations, éventuellement terminée par une instruction de branchement.

arc = branchement éventuellement pris selon une condition.

NB : une sorte d'**automate fini** avec *gardes* sur transitions, un bloc **initial** et des blocs **finaux** terminés par `return`.

Graphe de flot de contrôle (sur un exemple)

Définition d'un CFG (*Control Flow Graph*)

Représente le **corps** de chaque fonction par un **graphe orienté** où

nœud = **bloc** avec 1 unique point d'entrée sur une séquence d'affectations, éventuellement terminée par une instruction de branchement.

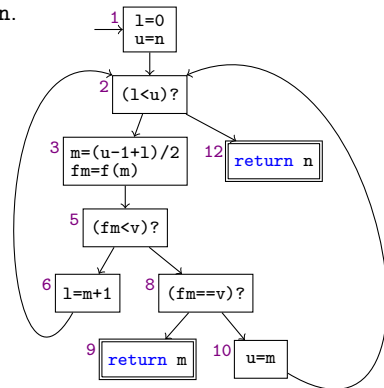
arc = branchement éventuellement pris selon une condition.

NB : une sorte d'**automate fini** avec *gardes* sur transitions, un bloc **initial** et des blocs **finaux** terminés par **return**.

```

1  unsigned int l=0, u=n;
2  while (l<u) {
3      unsigned int m=(u-1+1)/2;
4      long fm=f(m); //appel à f
5      if (fm<v) {
6          l=m+1;
7          continue; }
8      if (fm==v)
9          return m;
10     u=m;
11 }
12 return n;

```



Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle :
break, **continue** et **goto**

Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Quelques avantages des CFG

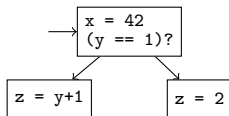
- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Exemples d'optimisations de CFG en compilation

Bout de CFG
initial

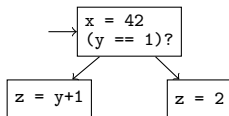


Quelques avantages des CFG

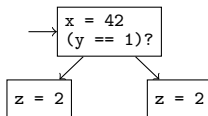
- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Exemples d'optimisations de CFG en compilation

Bout de CFG
initial



Après propag.
constante

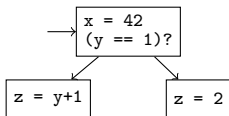


Quelques avantages des CFG

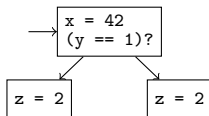
- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Exemples d'optimisations de CFG en compilation

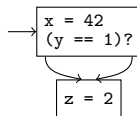
Bout de CFG
initial



Après propag.
constante



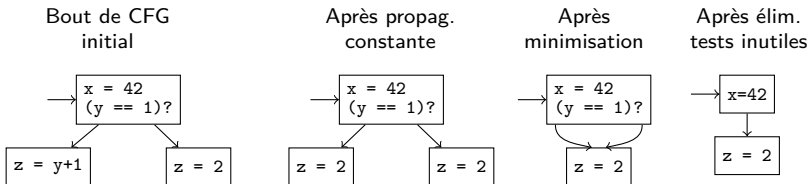
Après
minimisation



Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

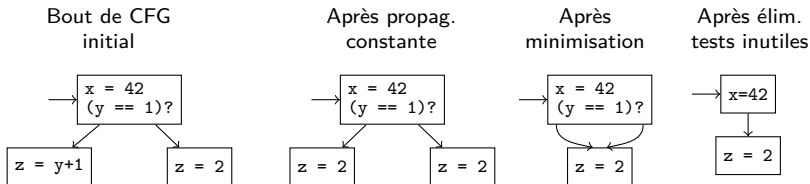
Exemples d'optimisations de CFG en compilation



Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Exemples d'optimisations de CFG en compilation

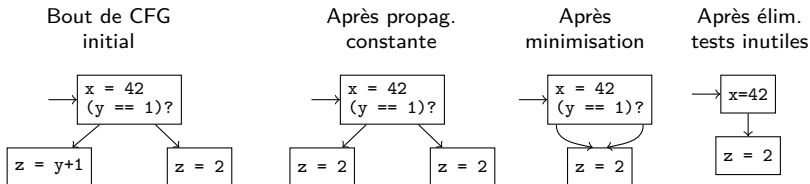


Comment intégrer “*élim. tests inutiles*” à analyse de liveness pour éviter certaines fausses dépendances, e.g. de `z` sur `y` ici ?

Quelques avantages des CFG

- ▶ Sémantique simple des instructions sur flot de contrôle : **break**, **continue** et **goto**
- ▶ Réutilisation d'algorithmes sur les **automates finis**, comme *minimisation* des automates déterministes

Exemples d'optimisations de CFG en compilation



Comment intégrer “*élim. tests inutiles*” à analyse de liveness pour éviter certaines fausses dépendances, e.g. de `z` sur `y` ici ?

réponse diapo 41

Encodage des CFG en commandes gardées

sur l'exemple de la diapo 38

Encodage de l'**automate** avec variable explicite de pc
contenant numéro du prochain bloc à exécuter

```
pc := 1;  
(  
  (assume pc=1; l := 0; u := n; pc := 2)  
  ⊓ (assume pc=2; if l<u then pc := 3 else pc := 12)  
  ⊓ (assume pc=3; m := (u-1+l)/2; fm ← f(m); pc := 5)  
  ⊓ (assume pc=5; if fm<v then pc := 6 else pc := 8)  
  ⊓ (assume pc=6; l := m+1; pc := 2)  
  ⊓ (assume pc=8; if fm==v then pc := 9 else pc := 10)  
  ⊓ (assume pc=9; return m)  
  ⊓ (assume pc=10; u := m; pc := 2)  
  ⊓ (assume pc=12; return n)  
)*
```

Encodage des CFG en commandes gardées

sur l'exemple de la diapo 38

Encodage de l'**automate** avec variable explicite de pc
contenant numéro du prochain bloc à exécuter

```
pc := 1;  
(  
  (assume pc=1; l := 0; u := n; pc := 2)  
  ⊓ (assume pc=2; if l<u then pc := 3 else pc := 12)  
  ⊓ (assume pc=3; m := (u-1+1)/2; fm ← f(m); pc := 5)  
  ⊓ (assume pc=5; if fm<v then pc := 6 else pc := 8)  
  ⊓ (assume pc=6; l := m+1; pc := 2)  
  ⊓ (assume pc=8; if fm==v then pc := 9 else pc := 10)  
  ⊓ (assume pc=9; return m)  
  ⊓ (assume pc=10; u := m; pc := 2)  
  ⊓ (assume pc=12; return n)  
)*
```

Permet **conditions sur pc** dans *logiques de Hoare*
et domaines abstraits !

Raffinement de la logique de Hoare avec liveness

Invariant I de boucle de forme $\bigwedge_i pc = i \Rightarrow I_i$
et $I[pc \leftarrow k] \Leftrightarrow I_k$

Raffinement de la logique de Hoare avec liveness

Invariant I de boucle de forme $\bigwedge_i pc = i \Rightarrow I_i$
 et $I[pc \leftarrow k] \Leftrightarrow I_k$

Règles pour affaiblir précondition de *live-in* dans gardes :

- ▶ pc pas “observé”

$$\frac{}{\langle pc = i \Rightarrow Q \rangle \mathbf{assume} \ pc = i \langle Q \rangle}$$

- ▶ **élimination des tests inutiles**

$$\frac{}{\langle I[pc \leftarrow i_1] \rangle \mathbf{if} \ C \ \mathbf{then} \ pc := i_1 \ \mathbf{else} \ pc := i_2 \langle I \rangle} \quad I[pc \leftarrow i_1] \Leftrightarrow I[pc \leftarrow i_2]$$

Raffinement de la logique de Hoare avec liveness

Invariant I de boucle de forme $\bigwedge_i pc = i \Rightarrow I_i$
 et $I[pc \leftarrow k] \Leftrightarrow I_k$

Règles pour affaiblir précondition de *live-in* dans gardes :

- ▶ pc pas “observé”

$$\frac{}{\langle pc = i \Rightarrow Q \rangle \mathbf{assume} \ pc = i \langle Q \rangle}$$

- ▶ **élimination des tests inutiles**

$$\frac{}{\langle I[pc \leftarrow i_1] \rangle \mathbf{if} \ C \ \mathbf{then} \ pc := i_1 \ \mathbf{else} \ pc := i_2 \langle I \rangle} \quad I[pc \leftarrow i_1] \Leftrightarrow I[pc \leftarrow i_2]$$

Exercice : adapter $\heartsuit \langle . \rangle S \langle . \rangle$ (cf. TD).

Conclusion vers quelques notions plus avancées

Sémantiques formelles axiomatiques (logiques de Hoare)
utiles comme *modèles simples* des analyses de programmes,

Conclusion vers quelques notions plus avancées

Sémantiques formelles axiomatiques (logiques de Hoare)

utiles comme *modèles simples* des analyses de programmes,
mais besoin de sémantiques formelles *plus fondamentales* :

- ▶ **sémantiques opérationnelles**, proches de l'intuition
- ▶ ou **sémantiques dénotationnelles**,
pour davantage de découplage entre
définitions sémantiques et représentations de programme ;

pour notamment, notion de **variable vivante**,

finement décrite via **non-interférence**

un concept sémantique qui *compare deux exécutions*.

Conclusion vers quelques notions plus avancées

Sémantiques formelles axiomatiques (logiques de Hoare)

utiles comme *modèles simples* des analyses de programmes,
mais besoin de sémantiques formelles *plus fondamentales* :

- ▶ **sémantiques opérationnelles**, proches de l'intuition
- ▶ ou **sémantiques dénotationnelles**,
pour davantage de découplage entre
définitions sémantiques et représentations de programme ;

pour notamment, notion de **variable vivante**,

finement décrite via **non-interférence**

un concept sémantique qui *compare deux exécutions*.

Importance des IR (représentations de programme)

CFG *améliorés* : **SSA (Single-Static-Assignment)**, cf. LLVM
car permet des états abstraits "*creux*" (*sparse*)