Spécification et vérification de programmes

17 mars 2025

Allan Blanchard

Partie I

Introduction générale

Pourquoi ce cours ?

Que va-t-on y faire ?

Développeurs, programmes et bugs

- Avez-vous déjà développé du logiciel ?

Développeurs, programmes et bugs

- Avez-vous déjà développé du logiciel ?
- Avez-vous déjà développé du logiciel contenant des bugs ?

Développeurs, programmes et bugs

- Avez-vous déjà développé du logiciel ?
- Avez-vous déjà développé du logiciel contenant des bugs ?
- De quelle nature étaient ces bugs ?

Développeurs, programmes et bugs

- Avez-vous déjà développé du logiciel ?
- Avez-vous déjà développé du logiciel contenant des bugs ?
- De quelle nature étaient ces bugs ?
- Avez-vous déjà développé du logiciel sans bugs ?

Développeurs, programmes et bugs

- Avez-vous déjà développé du logiciel ?
- Avez-vous déjà développé du logiciel contenant des bugs ?
- De quelle nature étaient ces bugs ?
- Avez-vous déjà développé du logiciel sans bugs ?
- Si oui, comment le savez-vous ?

Développeurs, programmes et bugs

- Avez-vous déjà développé du logiciel ?
- Avez-vous déjà développé du logiciel contenant des bugs ?
- De quelle nature étaient ces bugs ?
- Avez-vous déjà développé du logiciel sans bugs ?
- Si oui, comment le savez-vous ?

Tout programme non trivial contient (au moins) un bug (Loi de Murphy)

Anomalie logicielle

Comportement observé du logiciel qui diffère du comportement attendu

Propriétés fonctionnelles

- Crash
- Exception non rattrapée
- Résultat erroné

Propriétés non-fonctionnelles

- Mémoire consommée trop importante,
- Temps d'exécution trop important,
- Observation de données confidentielles
- Non-terminaison ? Non !

Toutes les anomalies ne se valent pas

Le caractère gênant des anomalies dépend

- de leur fréquence,
- de leur dangerosité,
- de la facilité à les contourner,
- de leur impact sur l'utilisabilité du logiciel,
- de leur impact sur l'environnement extérieur,
- ...

Ariane 5, 1996

- en 1996, destruction de la fusée Ariane 5 durant son vol inaugural
- Cause :
  - réutilisation d'un composant d'Ariane 4... sans respecter sa spécification
  - débordement lors de la conversion d'un float de 64 bits vers un int 16 bits
- Conséquence :
  - mise hors service du composant
  - mauvaise interprétation du pilote automatique
  - déviation rapide de trajectoire
  - arrachage des boosters
  - autodestruction préventive de la fusée
- Coût : $\approx$ 500 millions de dollars
- l'un des bugs les plus chers de l'histoire

Therac-25, 1985

- de 1985 à 1987, l'appareil de radiothérapie Therac-25 est responsable du décès de plusieurs personnes
- Causes :
  - la cause première était la mauvaise architecture du système et de mauvaises pratiques de développements plutôt que les erreurs de programmation trouvées,
  - en particulier, il n'était pas possible de tester automatiquement et proprement le système
- Conséquence :
  - l'utilisateur pouvait programmer un surdosage du patient sans s'en rendre compte
  - aucun dispositif de contrôle
  - les patients se retrouvaient alors sévèrement irradiés
- l'un des bugs les plus graves de l'histoire

Et plein d'autres...

http://en.wikipedia.org/wiki/List_of_software_bugs

Catégories

- Exploration spatiale
- Médical
- *tracking years*
- Réseau électrique
- Finance
- Télécommunication
- Militaire
- Média
- Jeux
- Cryptographie

L'évolution de la taille des systèmes

Nos systèmes tendent à devenir de plus en plus gros

Exemple : la taille de Linux

- 1991 : 10 kLOC
- 2001 : 2.4 MLOC
- 2003 : 5.2 MLOC
- 2009 : 11 MLOC
- 2015 : 20.2 MLOC
- 2020 : 27.8 MLOC

Problème de taille

- le nombre de bugs est fonction de la taille de système ...
- la complexité du système global aussi ...

Les erreurs coûtent cher

- coût estimé des bugs :
  - USA 2020 : 607 milliards de dollars (CPSQ report 2020)
- Coût pour l'utilisateur :
  - accidents graves, voir mortels,
  - dégâts environnementaux,
  - destruction de matériels.
- Coût pour le fournisseur :
  - coût du correctif,
  - coût du remboursement.

Confiance

Avoir confiance dans les logiciels est donc un enjeu sociétal et économique majeur.

Établir la confiance ?

- Idéalement, en garantissant qu'il n'a "aucune erreur"
- Qu'est-ce-que cela veut dire ?
  - Informellement : il fait bien ce qu'il est censé faire
  - Formellement : à démêler dans ce cours
- Cet idéal est-il possible ?

Établir la confiance ?

- Idéalement, en garantissant qu'il n'a "aucune erreur"
- Qu'est-ce-que cela veut dire ?
  - Informellement : il fait bien ce qu'il est censé faire
  - Formellement : à démêler dans ce cours
- Cet idéal est-il possible ? Non ! (plus de détails par la suite)
- En pratique :
  - en se préservant des erreurs "autant que possible"
  - en atteignant le rapport qualité/prix jugé optimal
- Comment définir l'optimal ?
  - Logiciel "normal" : défini par le client
  - Logiciel "critique" : imposé par des lois/normes/assurances

La DO-178C

Définition et utilité

- La norme DO-178 traite de la sûreté des logiciels avioniques.
- Guide pour déterminer si le logiciel sera fiable dans un environnement avionique.
- La suivre est requis pour certifier le logiciel pour l'utiliser en production.
- La version DO-178C de cette norme date de 2011.
- La première version de la DO-178 date de 1982.

(Informations de cette section extraites de Wikipédia, cette norme coûte un bras)

Design Assurance Level (DAL)

A ⟩ Catastrophique – morts multiples, perte de l'avion.

B ⟩ Dangereux – réduction importante de la sécurité, équipage dans l'incapacité d'accomplir ses tâches, blessures sérieuses ou mort de quelques occupants.

C ⟩ Majeur – réduction significative de la sécurité, équipage en difficulté, inconfort pour les occupants et risque de blessures.

D ⟩ Mineur – le problème est visible, mais n'implique pas de risque de sécurité.

E ⟩ Pas d'effet

Exigences

DAL	Objectives	Failure Rate
A	66	10^{-9}/h
B	65	10^{-7}/h
C	57	10^{-5}/h
D	28	10^{-3}/h
E	0	n/a

Acceptabilité des accidents catastrophiques

- Constat : 1 accident grave pour 10^6 heures de vol
- Constat : 10% liés à des défaillances des systèmes avioniques
- Conclusion : seuil acceptable d'accidents catastrophiques imputables aux systèmes = 10^{-7} par heure de vol
- arbitrairement, on considère que $\approx 10^2$ de conditions de pannes peuvent être catastrophiques et qu'elles sont équiprobables
- donc, probabilité acceptable d'une condition de panne catastrophique par heure de vol = 10^{-9}
- les contraintes imposées sur le cycle de développement sont fonction de cette probabilité

Niveau E

Aucune contrainte de développement

Niveau D

- logiciel documenté ; la liste des documents fixée par la norme
- des plans doivent être établis pour fixer les méthodes de développement, de vérification, de gestion de configuration, d'assurance qualité
- traçabilité spécification du système $\rightarrow$ spécifications de haut niveau $\rightarrow$ vérifications
- tout ce qui est spécifié doit être vérifié : couverture fonctionnelle et documents doivent aussi être vérifiés
- le logiciel doit être géré en configuration, toutes les évolutions du code source doivent être justifiées
- Un service Qualité indépendant doit assurer le respect de la norme

Niveau C

En plus des contraintes du niveau D :

- des règles de développement doivent être fixées (spécification, conception et codage)
- les contraintes de traçabilité et de vérification s'appliquent à la conception et au codage
- les exigences de bas niveau doivent être vérifiées
- la couverture structurelle, ou couverture de code, doit être vérifiée et analysée

Niveau B

En plus des contraintes du niveau C :

- La couverture de code au niveau “décision” est requise.
- Les activités de développement et de vérification doivent être confiées à des équipes indépendantes.

Niveau A

En plus des contraintes du niveau B :

- La couverture de code au niveau “condition”/“décision” est requise.

Vérifier et Valider (V & V)

Quelle est la différence entre les deux ?

Vérifier et Valider (V & V)

Quelle est la différence entre les deux ?

Vérification : "Are we building the product right ?"

Point de vue fournisseur : est-ce-que le logiciel fonctionne correctement ?

Validation : "Are we building the right product ?"

Point de vue utilisateur : est-ce-que le logiciel fait bien ce que l'on veut ?

Sémantique et spécification

- Besoin de définir ce que fait un programme, *i.e.* sa sémantique
- besoin de définir ce qu'est censé faire un programme, *i.e.* sa spécification
- comment garantir qu'un programme (caractérisé par sa sémantique) satisfait sa spécification ?

Analyse de programmes

Analyse de programmes

Déterminer certaines propriétés d'un programme, en particulier des propriétés sémantiques sur son comportement à l'exécution.

Dynamique

L'analyseur peut exécuter le programme

Statique

L'analyseur ne peut pas exécuter le programme

Analyse dynamique

peut exécuter le programme

- test (chapitre II)
- monitoring (chapitre IV)
- avantage : prend en compte le contexte d'exécution
- inconvénient : subit/influe sur le contexte d'exécution

observe le code du programme, mais ne l'exécute pas

Quel code ?

- Source (ici, C)
- Assembleur
- Binaire

Quelles analyses ?

- interprétation abstraite (chapitre V)
- méthode déductive (chapitre VI)
- inconvénient ? : ne prend pas en compte le contexte d'exécution, voire de compilation
- avantage : ne nécessite pas/n'influe pas sur le contexte d'exécution

Analyse statique pour l'optimisation (aparté)

Usage majeur historique des analyses statiques

Les analyses statiques sont très utilisées dans les compilateurs.

Exemples

- Propagation de constantes,
- élimination de sous-expressions communes,
- élimination de code mort,
- variables “vivantes” et allocation de registres...

- vérifier et valider un programme est long et compliqué
- vu la taille des systèmes actuels, impossible à faire manuellement

Besoin d'automatisation et d'outillage

L'outil parfait

Quel outil parfait pour garantir qu'un programme P satisfait sa spécification S ?

L'outil parfait

Quel outil parfait pour garantir qu'un programme P satisfait sa spécification S ?

1. Donner P et S en entrée à un outil O
2. appuyer sur "entrée" (analyse automatique)
3. O regarde P et S, idéalement sans exécuter P
4. O donne instantanément la réponse
   - oui : P satisfait S
   - non : il y a un problème à tel endroit P par rapport à tel point de S
5. être sûr que la réponse donnée par O est correcte

L'outil parfait

Quel outil parfait pour garantir qu'un programme P satisfait sa spécification S ?

1. Donner P et S en entrée à un outil O
2. appuyer sur "entrée" (analyse automatique)
3. O regarde P et S, idéalement sans exécuter P
4. O donne instantanément la réponse
   - oui : P satisfait S
   - non : il y a un problème à tel endroit P par rapport à tel point de S
5. être sûr que la réponse donnée par O est correcte

Problème : L'univers n'est pas indifférent à l'intelligence, il lui est activement hostile

Une analyse statique automatique exacte est impossible.

Théorème de Rice (1953)

Toute propriété

- extensionnelle (qui dépend uniquement de la sémantique du programme)
- non triviale (ni toujours vraie, ni toujours fausse)

est

- indécidable

Exemple : problème de l'arrêt

Il est impossible de déterminer statiquement si un programme quelconque termine (en temps fini) quelles que soient ses données d'entrée.

Contrecarrer le théorème de Rice

- Test : exécute le programme pour trouver des bugs. (Chapitre II)
- Model-checking : travailler non pas sur le programme, mais sur un modèle du programme (Non traité dans ce cours)
- Interprétation abstraite : effectuer des approximations de la sémantique concrète. (Chapitre V)
- Méthodes déductives : raisonner de manière précise, par déduction, sur le programme et extraire les problèmes “durs” (Chapitre VI)

Chaque technique a ses avantages...
Et au moins un gros inconvénient

Point d'étape

- Que cherche-t-on à faire ?

Point d'étape

- Que cherche-t-on à faire ?
  - Vérifier : est-ce-que le logiciel fonctionne correctement ?
  - Valider : est-ce-que le logiciel fait bien ce qu'il est censé faire ?
- Comment cherche-t-on à le faire ?

Point d'étape

- Que cherche-t-on à faire ?
  - Vérifier : est-ce-que le logiciel fonctionne correctement ?
  - Valider : est-ce-que le logiciel fait bien ce qu'il est censé faire ?
- Comment cherche-t-on à le faire ?
  - En analysant le programmes
  - En outillant ce processus d'analyse pour qu'il se rapproche de l'idéal :
    - correct
    - exact
    - automatique (et instantané)
    - statique

Menu

Outillage pour ce cours

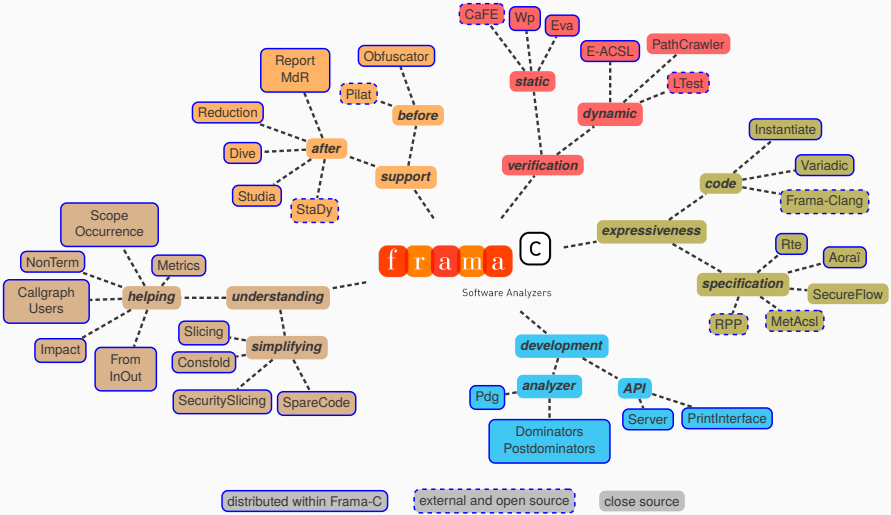
- C
  - le langage est tout tordu et plein de pièges
- utest
  - un outil de test header-only
- gcovr
  - un outil d'analyse de couverture pour GCC
- Frama-C
  - une suite d'outils pour l'analyse de code C

Frama-C

Plateforme dédiée à l'analyse de code C

- développée depuis 2004,
- utilisée industriellement (THALES, Airbus, ANSSI, ...),
- open-source,
- extensible, basée sur une architecture à plug-ins,
- principalement utilisée pour de l'analyse formelle.

FRAmework for Modular Analysis of C code



Partie II

Test de programmes

Comment tester ?

Comment évaluer des tests ?

Votre expérience de testeurs

- Avez-vous déjà testé un logiciel ?

Votre expérience de testeurs

- Avez-vous déjà testé un logiciel ?
- Était-ce le vôtre ou celui d'un tiers ?

Votre expérience de testeurs

- Avez-vous déjà testé un logiciel ?
- Était-ce le vôtre ou celui d'un tiers ?
- Quel(s) étai(en)t votre/vos but(s) ?

Votre expérience de testeurs

- Avez-vous déjà testé un logiciel ?
- Était-ce le vôtre ou celui d'un tiers ?
- Quel(s) étai(en)t votre/vos but(s) ?
- Comment vous y preniez-vous ?
  - Au hasard ?
  - Avec une méthodologie particulière ? Laquelle ?

Votre expérience de testeurs

- Avez-vous déjà testé un logiciel ?
- Était-ce le vôtre ou celui d'un tiers ?
- Quel(s) étai(en)t votre/vos but(s) ?
- Comment vous y preniez-vous ?
  - Au hasard ?
  - Avec une méthodologie particulière ? Laquelle ?
- Combien de temps y avez-vous consacré (comparativement au reste du développement) ?

Qu'est-ce-que le test ?

Selon G. Myers

> *Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts.* *(*G. Myers (The Art of Software Testing, 1979)*)*

Selon l'IEEE

Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.

Qu'est-ce-que le test ?

Selon G. Myers

Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts. (G. Myers (The Art of Software Testing, 1979))

Selon l'IEEE

Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.

Ce que le test n'est pas

Testing can only reveal the presence of errors but never their absence.
(E. W. Dijkstra (Notes on Structured Programming, 1972))

Coût du test

- marché de plus de 40 milliards d'euros en 2020
- estimation à plus de 60 milliards d'ici à 2027
- $>$ 50% du temps du cycle de développement d'un logiciel critique (parfois $>$ 90%)
- en moyenne 30% pour un logiciel "standard"

Le test coûte cher

Coût du bug

Plus un bug est corrigé tard, plus le correctif est cher, empiriquement :

en phase de développement	coût 1
en phase d'intégration (bug de conception)	coût 10
en phase de recette (bug de spécification)	coût 100
en phase d'exploitation	coût $>$ 1000

Nécessité de détecter les problèmes au plus tôt du cycle de développement

Un bug finit par couter très cher

Deux aspects du test

1. Construire la qualité du produit
   - lors de la phase de conception, du codage,
   - en partie par les développeurs,
   - objectif : trouver rapidement le plus de bugs possible.
2. Démontrer la qualité du produit à un tiers
   - une fois le produit terminé,
   - idéalement : par une équipe dédiée,
   - objectif : convaincre un organisme de certification, le client ...

Cas de test

Scénario à exécuter pour tout ou partie du système, composition :

- caractéristiques de l'entrée correspondante,
- estimation de l'oracle, le résultat attendu.

On dérive :

- une donnée de test représentative de l'entrée attendue,
- un oracle concret correspondant.

Finalement :

- on exécute le test,
- on compare* le résultat avec l'oracle concret (Verdict : succès/échec)

* Remarque : l'égalité stricte n'est pas forcément nécessaire

Oracle

Définir l'oracle est un problème difficile

"Bons" cas

- le programme ne plante pas
- programmation par contrat
- test dos à dos : tester avec un logiciel similaire

Processus de test

Suite et script

- Suite de tests : ensemble de tests du système
- Script de test : code qui lance les différents tests et collecte les résultats

Test de programme

Analyse dynamique !

Bien tester ?

Le but des tests est de trouver des bugs

Ne pas tester en supposant qu'aucun bug ne va être trouvé !

Quelques guidelines sur l'attitude à adopter

- Cherchez les bugs
- Définissez les sorties voulues avant de lancer le test
- Observez méticuleusement les oracles
- Idéalement : le développement et le test sont faits par deux personnes différentes

Une bonne suite de tests

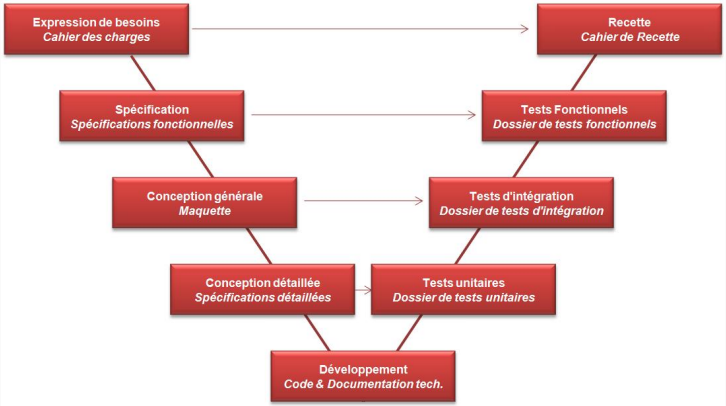
- exercer un maximum de comportements différents du programme
- tests nominaux
  - cas de fonctionnements les plus fréquents
  - forte probabilité de survenir
  - peu de risque d'erreurs
- tests de robustesse
  - cas de fonctionnements les moins fréquents
  - faible probabilité de survenir
  - risque d'erreurs important

Très difficile à définir en pratique

Types de test

- test unitaire : tester les différents composants en isolation,
- test d'intégration : tester après composition des composants,
- test système/de conformité : valider globalement le code
  - inclut si nécessaire les tests de sécurité, de performance, ...
- test de validation/acceptation : valider l'adéquation aux besoins du client
  - similaire au test système, mais réalisé/vérifié par le client
- test de non-régression : vérifier que les corrections/évolutions du code n'ont pas introduit de bugs

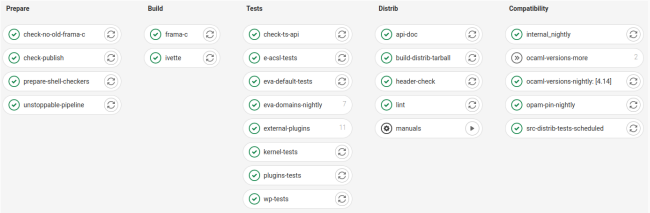
Cycle en V



- sans oublier la maintenance et les tests de non-régression
- dans des processus plus mainstream, on répète des V courts

Tester tout le temps

- chaque ajout de code est testé avant intégration
- chaque intégration est re-validée a posteriori
- des tests plus poussés peuvent être joués à intervalles réguliers



Bénéfices

- pas de travail ajouté pour le développeur,
- détection au plus tôt des problèmes

Test-driven development (TDD)

Méthodologie

1. écrire un seul test qui décrit une partie du problème à résoudre,
2. vérifier que le test échoue (le code qui permet sa réussite n'existe pas encore),
3. écrire juste assez de code pour que le test réussisse,
4. vérifier que le test passe, ainsi que les autres tests existants,
5. remanier le code/le test sans altérer le comportement,
6. revenir à 1 si la fonctionnalité n'est pas encore complète.

Bénéfices

- Permet d'avoir rapidement de nombreux tests de bonne qualité
- Construit la bonne qualité du logiciel pendant le développement

Tester chaque composant en isolation

- généralement la granularité est au niveau de la fonction

Approches

- top-down (depuis le point d'entrée) ou bottom-up (depuis les feuilles)
- dans la réalité, souvent un mix des deux au cas par cas

Code incomplet

- code appelé manquant : création d'une *doublure*
- code appelant manquant : création d'un lanceur

Comparaisons des approches

- top-down
  - ✗ Les *doublures* peuvent être complexes à écrire
  - ✓ On teste beaucoup plus les composants de haut niveau
- bottom-up
  - ✓ Les lanceurs sont généralement assez facile écrire
  - ✗ On risque de moins tester les composants de haut niveau

Les différentes sortes de doublures

- Dummy : nécessaire au test (*e.g.* paramètre) mais pas utilisé
- Stub : utilisée par le test, mais a une liste fixe de retours
- Spy : utilisée par le test, observe cet usage (pour contrôle *a posteriori*)
- Mock : utilisée par le test, observe cet usage (avec contrôle *immédiat*)
- Fake : répond à la spécification du code remplacé, mais simplifié

Plus la doublure est avancée, plus elle est difficile à produire et fragilise le test

Liste non exhaustive

- Test exhaustif : on teste toutes les entrées (impossible en pratique)
- Test probabiliste : on teste à partir de données statistiques
- Test fonctionnel : test boîte noire, à partir des spécifications, sans regard sur le code
- Test structurel : test boîte blanche, en prenant en compte la structure du code

Test probabiliste

Test aléatoire : loi uniforme

- ✓ Test massif facile (si les oracles sont automatisés)
- ✓ Pas de biais statistique
- ✗ Très difficile de générer des comportements spécifiques

Test statistique : loi dépendant de la cible

- › Biais statistique important
  - ✓ On teste beaucoup les comportements communs
  - ✗ Mais qui sont généralement ceux qui ne posent pas de problèmes
- ✗ Loi statistique généralement dure à produire

Variante probabiliste : le cas du fuzzing

Fuzz Testing

- génération automatique de données aléatoires en entrée,
- algorithmes plus ou moins malins pour explorer le comportement du programme,
- très efficace pour détecter des plantages ou erreurs critiques,
- particulièrement apprécié pour les questions de sécurité

Test fonctionnel (boîte noire)

Basé sur les entrées du composant testé

- ✓ aucune connaissance interne du composant nécessaire
- ✓ bien adapté pour vérifier la conformité par rapport à la spécification
- ✗ difficile de détecter des défauts précis de programmation
- ✗ difficile de produire une aide automatisée pour l'évaluation de ces tests

Bien adapté du niveau unitaire à système *

*En particulier, à haut niveau, le binaire est suffisant pour tester

Test structurel (boîte blanche)

Basé sur la structure du code

- ✗ nécessite le code
- ✓ le code est plus précis que la spécification
- ✗ le code est généralement gros par rapport à la spécification
- ✓ tests plus précis
- ✗ ? tests plus nombreux
- ✓ efficace pour détecter les défauts précis de programmation
- ✗ peu adapté pour détecter le non-respect d'une spécification

Couvrir le domaine d'entrée

Principe

- diviser le domaine d'entrée en un nombre fini de classes
- une valeur à tester par classe uniquement

Couvrir le domaine d'entrée

Principe

- diviser le domaine d'entrée en un nombre fini de classes
- une valeur à tester par classe uniquement

Exemple : valeur absolue – `int abs(int);`

- trois classes d'entrée "naturelles" : < 0, $= 0$, > 0,
- on sélectionne une valeur dans chaque classe (*e.g.* -42, 0, 24).

Deux grands types de partitions

Basée sur l'interface

- Basée uniquement sur les types des données d'entrée
- Assez facile à automatiser

Basée sur les fonctionnalités

- Prend en compte les relations entre les variables d'entrée
- Plus pertinent
- Peu automatisable

Exemple : classes pour la recherche

```
bool search(int v, list<int> l);
```

Interface

- $\{empty(l), \neg empty(l)\}$
- $\{< 0, = 0, > 0\}$

Fonctionalités

- $\{empty(l), v \in l, \neg empty(l) \wedge e \notin l\}$

Différentes situations

- API interne : pas de programmation défensive
  - généralement peu d'intérêt aux tests invalides
- API externe : programmation défensive (parfois aidée par le typage)
  - tests invalides nécessaires
- Interface utilisateur : programmation défensive
  - tests invalides nécessaires

Différentes situations

- API interne : pas de programmation défensive
  - généralement peu d'intérêt aux tests invalides
- API externe : programmation défensive (parfois aidée par le typage)
  - tests invalides nécessaires
- Interface utilisateur : programmation défensive
  - tests invalides nécessaires

Conseil : Attention à en faire !

Cas d'invalidité

Différentes situations

- API interne : pas de programmation défensive
  - généralement peu d'intérêt aux tests invalides
- API externe : programmation défensive (parfois aidée par le typage)
  - tests invalides nécessaires
- Interface utilisateur : programmation défensive
  - tests invalides nécessaires

Conseil : Attention à en faire !
Conseil : Attention à ne pas trop en faire !

Tactique pour améliorer l'efficacité des jeux de données

Les erreurs se produisent souvent sur les cas limites, testons ces cas

- s'intègre très naturellement aux tests basés sur les classes
- génère plus de blocs, donc plus cher
- on teste les bornes des classes d'équivalence

Exemples

- sur une classe $[N..M]$, tester N, $N+1$, $M-1$, et M
- ensemble vide, singleton
- fichier vide, fichier de taille max, fichier juste trop gros,
- ...

Graphe de contrôle

À un programme P, on associe un graphe de contrôle étiqueté $(\mathcal{S}, \mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{F})$, où :

- À chaque instruction de P on associe un nœud de $\mathcal{S}$.
- $\mathcal{T} \subset (\mathcal{S} \times \mathcal{S})$ est l'ensemble des arcs
- $\mathcal{E} : \mathcal{S} \mapsto \{x = e \mid ?e \mid \texttt{skip}\}$ associe une étiquette à chaque arc.
- $\mathcal{I} \in \mathcal{S}$ représente l'état initial
- $\mathcal{F} \in \mathcal{S}$ représente l'état final

Exemple de graphe de flot de contrôle

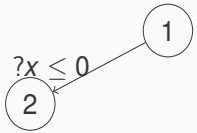
```
if(x <= 0){              // 1
  x = 1                  // 2
} else {
  ;                      // 3
}
y = 1;                   // 4
while (! (x <= 0)) {     // 5
  y = y * x;             // 6
  x--;                   // 7
}
```

Exemple de graphe de flot de contrôle

```
if(x <= 0){              // 1
  x = 1                  // 2
} else {
  ;                      // 3
}
y = 1;                   // 4
while (! (x <= 0)) {     // 5
  y = y * x;             // 6
  x--;                   // 7
}
```

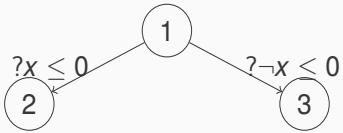
Exemple de graphe de flot de contrôle

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



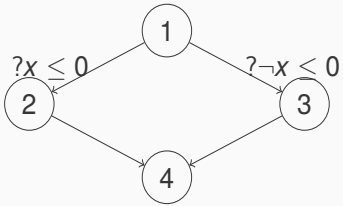
Exemple de graphe de flot de contrôle

```
if(x <= 0){              // 1
  x = 1                  // 2
} else {
  ;                      // 3
}
y = 1;                   // 4
while (! (x <= 0)) {     // 5
  y = y * x;             // 6
  x--;                   // 7
}
```



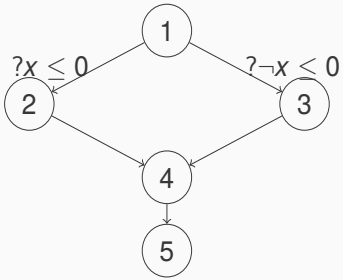
Exemple de graphe de flot de contrôle

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



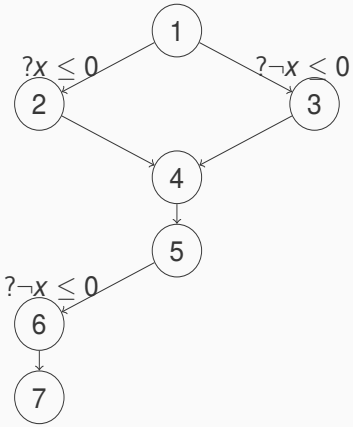
Exemple de graphe de flot de contrôle

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



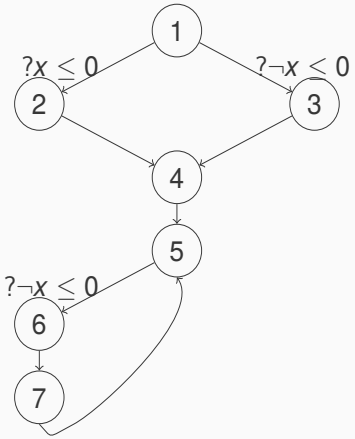
Exemple de graphe de flot de contrôle

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



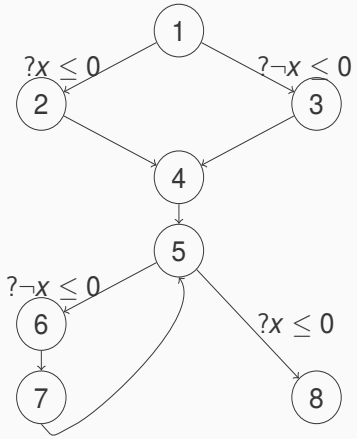
Exemple de graphe de flot de contrôle

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



Exemple de graphe de flot de contrôle

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



Critères de couverture du flot de contrôle

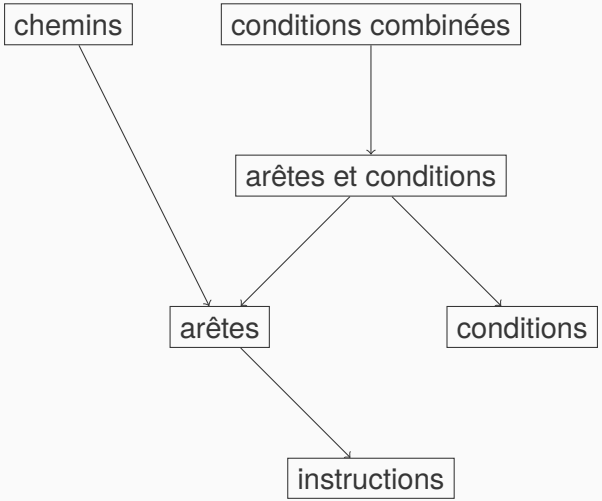
- Instructions : chaque instruction exécutable (nœud), exceptée `skip`, du programme apparaît dans un cas de test.
- Arêtes : chaque arête exécutable (exceptée `skip`) du graphe de contrôle apparaît dans un cas de test.
- Conditions : chaque prédicat atomique du programme apparaît dans un cas de test, évalué à vrai et une autre évalué à faux.
- Arêtes et conditions : couverture des arêtes + couverture des conditions
- Couverture des conditions combinées : similaire à la couverture des conditions, mais toutes les combinaisons de prédicats interprétés à `vrai` et `faux` doivent être couvertes

Couverture des chemins

Chaque chemin exécutable du graphe de contrôle apparaît dans un cas de test.

- Boucles dont le nombre d'itérations est inconnu !
  - Couverture des k-chemins : pour chaque boucle, tous les chemins contenant i répétitions de la boucle, $0 \leq i \leq k$.
  - Couverture PLCS (Portion Linéaire de Code suivie d'un Saut) : un saut est l'entrée, la sortie ou l'arrivée d'un branchement.
- Caractère indécidable des chemins/instructions/arêtes exécutable requis par certains critères
  - la présence de code non exécutable est souvent le signe d'un code mal écrit, voire faux
  - interdit dans les programmes critiques

Hiérarchie des critères de couverture structurelle



Flot de données

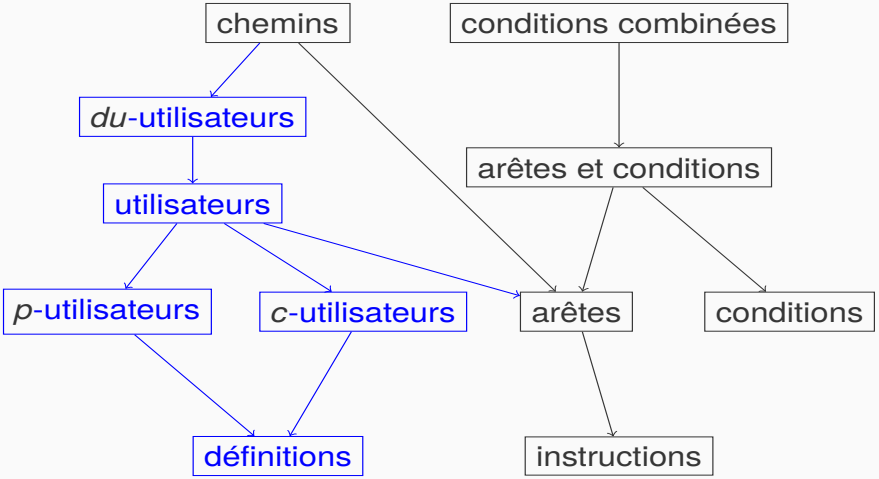
Les critères fondés sur le flot de contrôle ne sont pas assez fins

- variable définie : lorsque sa valeur est modifiée
- variable utilisée :
  - *p*-utilisée : lorsque sa valeur est lue dans un prédicat de branchement
  - *c*-utilisée : dans les autres cas (valeur lue pour un calcul)
- une instruction *i* est utilisatrice d'une variable *x* par rapport à une instruction de définition *j* si :
  - *x* est définie en *j*
  - *x* est utilisée en *i*
  - il existe un chemin sur lequel *x* n'est pas redéfinie entre *i* et *j*

Critères de couverture du flot de données

- Couverture toutes les définitions : pour chaque définition, un chemin utilisateur apparaît dans un cas de test.
- Couverture tous les *p*-utilisateurs : pour chaque définition, chaque instruction *p*-utilisatrice apparaît dans un cas de test.
- Couverture tous les *c*-utilisateurs : pour chaque définition, chaque instruction *c*-utilisatrice apparaît dans un cas de test.
- Couverture tous les utilisateurs : *p*-utilisateurs + *c*-utilisateurs.
- Couverture toutes les *du*-utilisateurs : pour chaque définition *d*, tous les chemins entre *d* et chacune de ses instructions utilisatrices sont couverts.

Hiérarchie des critères de couverture structurelle



D'autres critères de couverture existent

Dilemmes du testeur

Que déduire si la campagne de tests trouve peu d'erreurs ?

- optimiste : le programme est très bon
- pessimiste : les tests sont mauvais

Que déduire si la campagne de tests trouve beaucoup d'erreurs

- optimiste : la majorité des erreurs ont été corrigées
- pessimiste : le programme est de mauvaise qualité : il reste beaucoup d'erreurs

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ?

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire
- tous les tests passent ?

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire
- tous les tests passent ? Mauvaise idée : aucun intérêt de trouver des bugs

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire
- tous les tests passent ? Mauvaise idée : aucun intérêt de trouver des bugs
- (presque) plus de bugs trouvés ?

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire
- tous les tests passent ? Mauvaise idée : aucun intérêt de trouver des bugs
- (presque) plus de bugs trouvés ? Fausse bonne idée : effets de saturation

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire
- tous les tests passent ? Mauvaise idée : aucun intérêt de trouver des bugs
- (presque) plus de bugs trouvés ? Fausse bonne idée : effets de saturation
- quand le critère fixé est atteint ?

Quand arrêter une campagne de tests ?

- À la fin du budget alloué ? Mauvaise idée : autant ne rien faire
- tous les tests passent ? Mauvaise idée : aucun intérêt de trouver des bugs
- (presque) plus de bugs trouvés ? Fausse bonne idée : effets de saturation
- quand le critère fixé est atteint ? Bonne idée, mais :
  - attention à ne pas chercher à atteindre le critère plutôt qu'à trouver des bugs
  - distinguer objectifs et moyens

Effet de saturation

- nombre d'erreurs trouvées ne croît pas linéairement avec le temps
- 2 phases :
  - augmentation du nombre d'erreurs trouvées puis atteinte d'un palier : effet de seuil
  - stagnation du nombre d'erreurs trouvées
- conséquences :
  - ne pas arrêter les tests en période d'augmentation du nombre d'erreurs trouvées
  - ne pas arrêter les tests dès le premier palier

Critère d'arrêt en pratique

Proposition réaliste (?) de G. J. Myers

- constat 1 : le but est de trouver des bugs
- constat 2 : il y a toujours des bugs
- conséquence : exprimer le critère d'arrêt en fonction du nombre de bugs trouvés

Critère d'arrêt en pratique

- programme de 10 kLOC
- estimation de 50 erreurs par kLOC, donc 500 erreurs
- supposition de 200 erreurs de codage et 300 erreurs de conception
- volonté de trouver 98% des premières et 95% des secondes
- donc : trouver 196 erreurs de codage et 285 erreurs de conception
- critère d'arrêt choisi :
  - test unitaire : 130 erreurs trouvées (65%)
  - test d'intégration : 240 trouvées (30% de 200 et 60% de 300) ou 4 mois écoulés
  - test système : 111 erreurs trouvées ou 3 mois écoulés

Résumé

“Testing can only reveal the presence of errors but never their absence.”
*(*E. W. Dijkstra, Notes on Structured Programming, 1972*)*

- Correspond aux besoins réels de beaucoup d’industriels,
- Activité “ancienne”
  - bien intégrée aux cycles de développements actuels
  - ne nécessite pas de modifier les processus industriels
  - ne nécessite pas de modifier les équipes
  - existence de personnes qualifiées sur le marché
- activité simple
- le retour sur investissement proportionnel aux efforts
- peut trouver des erreurs indépendamment de celles cherchées

Bibliographie

- Glenford J. Myers The Art of Software Testing
  J. Willey & Sons, 1979
- Boris Beizer Software Testing Techniques
  Van Nostrand Reinhold, 1990 (seconde édition)
- S. Xanthakis, M. Maurice, A. de Amescua, O. Houri, L. Griffet
  Test & Contrôle des Logiciels
  Edition EC2, 1994
- S. Xanthakis, P. Régnier, C. Karapoulios Le test des logiciels
  Edition Hermes Sciences, 1999
- Aditya P. Mathur Foundations of Software Testing
  Edition Pearson Education, 2008

Partie III

Spécification

Comment spécifier ?

Différents types de spécification

Exprimer ce que l'on attend du programme

Différence fondamentale entre programme et spécification

- la spécification est le “quoi”, ce que l'on veut du système,
- le programme est le “comment”, ce que fait le système pour accomplir le “quoi”

La spécification est :

- le point de départ pour le développement
- le point de départ pour la vérification

Spécification informelle

Comment ?

- documents de spécification
- documentation du code

Avantages et inconvénients

- ✓ indispensables
- ✓ les moins difficiles à produire
- ✓ les plus facilement compréhensibles
- ✗ sources d'ambigüité, d'incohérence, d'incorrection, d'incomplétude,
- ✗ incompréhensibles par les outils

Nécessaire

Spécification semi-formelle

Comment ?

- diagrammes UML
- OCL (Object Constraint Language), le langage logique d'UML

Avantages et inconvénients

- ✓ permettent de spécifier facilement le modèle d'une application
- ✓ permettent de générer automatiquement le code trivial
- ✓ existence de pas mal outils facilitant leurs déploiements
- ✗ mêmes problèmes que les langues naturelles, mais de manière plus limitée

Souvent suffisant

Comment ?

- fondée sur des formalismes mathématiques/logiques solides

Avantages et inconvénients

- ✓ augmentent la compréhension et la confiance dans le code spécifié
- ✓ possibilité de vérifier des propriétés à propos des spécifications
- ✓ possibilité de prouver qu'un programme satisfait sa spécification
- ✗ nécessite de connaître et comprendre les formalismes sous-jacents
- ✗ problèmes des spécifications erronées, mais faibles risques

Nécessaire pour un code critique

Caractère exécutable d'une spécification

Définition

Une spécification est exécutable lorsque l'on peut produire un code qui permet sa vérification pendant l'exécution.

Le langage de spécification doit être adapté

- ✗ logique généralement moins expressive,
- ✓ possibilité de vérifier des propriétés à runtime.

Composants :

- variables,
- prédicats/relations à propos des variables,
- connecteurs logiques ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$),
- quantificateurs ($\exists$, $\forall$)

Tables de vérités

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
F	F	V	F	F	V	V
F	V	V	F	V	V	F
V	F	F	F	V	F	F
V	V	F	V	V	V	V

Quantificateurs

$\forall x.P(x)$

- “pour tout x, P(x) est vraie”
- ex : $\forall n : nat, n + 1 > n$

$\exists x.P(x)$

- “il existe un x tel que P(x) est vraie”
- ex : $\exists n : nat, n > 0$

Erreur commune avec l’existentielle et l’implication !

$$\exists x, P(x) \Rightarrow Q(x)$$

ACSL : ANSI/ISO C Specification Language

Présentation

- langage de spécification pour le langage C
- basé sur la notion de contrat
- permet de spécifier des propriétés fonctionnelles de programmes

Composants basiques

- une logique du premier ordre typée,
- expressions C pures,
- types du C + $\mathbb{Z}$ + $\mathbb{R}$
- prédicats et fonctions builtins

Parler de la mémoire en ACSL

C et la mémoire

- les pointeurs sont omniprésents en C,
- et sont causes d'erreur,
- besoin important d'une manière de modéliser cela.

Builtins ACSL

- `\valid(ptr),\valid(ptr+(0..n))`
- `\separated(ptr, other+(0..m))`
- `\initialized(ptr)`
- ...

Approche par contrats

- But : spécification de fonctions
- Approche : indiquer des assertions à propos des fonctions
  - Précondition supposée vraie à l'entrée (garanti par l'appelant)
  - Postcondition doit être vraie en sortie (garanti par la fonction)
- quand la précondition est fausse, rien n'est garanti

But d'un contrat

Refléter la spécification informelle

Annotations au sein du code :

```
/*@
  requires ... ;
  ensures  ... ;
*/
void function( ... ){ ... }
```

- introduction d'une annotation : `/*@` (ou `//@`),
- introduction d'une précondition : **`requires`**,
- introduction d'une postcondition : **`ensures`**.

Une partition des fonctionnalités : les comportements

```
/*@
  behavior <name>:
    assumes  ... ;
    requires ... ;
    ensures  ... ;
  ...
  disjoint behaviors ;
  complete behaviors ;
*/
void function( ... ){ ... }
```

- reflet du besoin de partitionner la spécification
- permet justement de vérifier la complétude et la disjonction des cas

Spécifier une propriété ponctuelle

```
int main(void){
  int x = 42 ;
  //@ assert x == 42 ;
}
```

Valeur absolue

```
int abs(int x){
  if(x < 0) return -x;
  return x ;
}
```

Valeur absolue (première spécification)

```
/*@ ensures x <  0 ==> \result == -x ;
    ensures x >= 0 ==> \result ==  x ;
*/
int abs(int x){
  if(x < 0) return -x;
  return x ;
}
```

Valeur absolue (comportements)

```
/*@ behavior negative:
      assumes x <  0 ;
      ensures \result == -x ;
    behavior positive:
      assumes x >= 0 ;
      ensures \result ==  x ;

    disjoint behaviors ;
    complete behaviors ;
*/
int abs(int x){
  if(x < 0) return -x;
  return x ;
}
```

Valeur absolue (logique built-in)

```
/*@ ensures \result == \abs(x) ; */
int abs(int x){
  if(x < 0) return -x;
  return x ;
}
```

Valeur absolue (logique built-in)

```
/*@ ensures \result == \abs(x) ; */
int abs(int x){
  if(x < 0) return -x;
  return x ;
}
```

La spécification n'est peut-être pas parfaite ceci dit, on verra ...

Recherche

```
size_t search(int v, int* arr, size_t l){
  for(size_t i = 0; i < l; i++)
    if(arr[i] == v) return i;
  return l;
}
```

Recherche (spécifiée)

```
/*@ behavior in:
      assumes \exists size_t o ; 0 <= o < l &&  arr[o] == v;
      ensures 0 <= \result < l && arr[\result] == v;
    behavior notin:
      assumes \forall size_t o ; 0 <= o < l ==> arr[o] != v;
      ensures \result == l;
    disjoint behaviors;
    complete behaviors;
*/ // Complete ? Ou pas ? On verra
size_t search(int v, int* arr, size_t l);
```

Recherche dichotomique

```
size_t bsearch(int* arr, size_t l, int value){
  if(l == 0) return 0 ;

  size_t low = 0 ;
  size_t up = l ;

  while(low < up){
    size_t mid = low + (up - low)/2 ;
    if      (arr[mid] > value) up = mid ;
    else if(arr[mid] < value) low = mid+1 ;
    else return mid ;
  }
  return l ;
}
```

Recherche dichotomique

```
/*@ requires
      \forall integer i, j ;
        0 <= i <= j < l ==> arr[i] <= arr[j] ;
    ...
*/
size_t bsearch(int* arr, size_t l, int value);
```

Quelques remarques

- la spécification peut être longue par rapport au code
- la spécification peut correspondre à plusieurs codes
- quid du caractère exécutable d'ACSL ?

Bibliographie : langages de spécification

- Eiffel : Analysis, Design and Programming Language 2nd edition. 2006.
  https://se.inf.ethz.ch/~meyer/ongoing/etl/EIFFEL-STANDARD.pdf
- Java Modeling Language (JML). 2000.
  http://www.eecs.ucf.edu/~leavens/JML/index.shtml
- Spec♯. 2004.
  http://research.microsoft.com/en-us/projects/specsharp
- ANSI C Specification Language (ACSL).
  https://frama-c.com/html/acsl.html
- Executable ANSI C Specification Language (E-ACSL).
  https://frama-c.com/fc-plugins/e-acsl.html
- Spark-2014.
  https://docs.adacore.com/spark2014-docs/html/lrm/

Partie IV

Vérification à l'exécution

Spécification exécutable ?

Comment générer le code correspondant à une spécification ?

Transformation une annotation en code

Comment transformer cette annotation ?

```
//@ assert a > INT_MIN;
```

Transformation une annotation en code

Comment transformer cette annotation ?

```
//@ assert a > INT_MIN;
```

```
assert (a > INT_MIN);
```

Est-ce toujours si simple ?

Les erreurs à l'exécution

Comment transformer cette annotation ?

```
//@ assert (a-b) / c > INT_MIN;
```

Les erreurs à l'exécution

Comment transformer cette annotation ?

```
//@ assert (a-b) / c > INT_MIN;
```

```
assert ((a-b) / c > INT_MIN);
```

Les erreurs à l'exécution

Comment transformer cette annotation ?

```
//@ assert (a-b) / c > INT_MIN;
```

```
assert ((a-b) / c > INT_MIN);
```

Non ! Risque de division par 0, de débordement ...

Les transformations ne doivent pas introduire de bugs !

Petit aparté ...

Combien de comportements indéterminés existent dans la norme ISO/C ?

Combien de comportements indéterminés existent dans la norme ISO/C ?

191

Combien de comportements indéterminés existent dans la norme ISO/C ?

191

Simple — Subtle

Modulo 0 — Strict aliasing violation

Shift with negative exponent — Data-race

Instrumentation automatique des erreurs à l'exécution

Accès invalide

```
//@ assert \valid(p+i) ;
int x = p[i] ;
```

Division par zéro

```
//@ assert d != 0 ;
int x = 42 / d ;
```

Débordement arithmétique

```
//@ assert INT_MIN <= a+b;
//@ assert a+b <= INT_MAX;
int x = a + b ;
```

- Initialisation,
- **float** vers **int**,
- *shift* hors bornes,
- ...

Ne pas générer de comportement indéfini

```
assert((long)a - b <= INT_MAX);
//@ assert a - b <= INT_MAX ;
assert((long)a - b >= INT_MIN);
//@ assert a - b >= INT_MIN ;
assert(c != 0);
//@ assert c != 0
assert((a-b) / c > INT_MIN);
//@ assert (a-b) / c > INT_MIN;
```

Méthode

- On génère une ligne,
- On l'analyse avec Rte,
- On génère (ou pas) une alarme,
- Et ainsi de suite.

Et la mémoire ?

Comment transformer cette annotation ?

```
//@ assert \valid(p+(0..i)) ;
```

Et la mémoire ?

Comment transformer cette annotation ?

```
//@ assert \valid(p+(0..i)) ;
```

Pas de solution simple

- il faut modéliser la mémoire à l'exécution,
- contrôler que tous les accès sont corrects,
- coût potentiellement important à l'exécution

Pour tout

Comment transformer cette annotation ?

```
//@ assert \forall integer i ; 0 <= i <= 42 ==> P(i);
```

Pour tout

Comment transformer cette annotation ?

```
//@ assert \forall integer i ; 0 <= i <= 42 ==> P(i);
```

```
for(int i = 0 ; i <= 42 ; ++i){
  assert(translated_P(i));
}
```

Il existe

Comment transformer cette annotation ?

```
//@ assert \exists integer i ; 0 <= i <= 42 && P(i);
```

Comment transformer cette annotation ?

```
//@ assert \exists integer i ; 0 <= i <= 42 && P(i);
```

```
int i = 0 ;
while(i <= 42){
  if(translated_P(i)) break;
  i++;
}
assert(i <= 42);
```

Caractère non exécutable des quantificateurs

Comment transformer cette annotation ?

```
//@ assert \forall integer i ; P(i);
//@ assert \exists integer i ; P(i);
```

Caractère non exécutable des quantificateurs

Comment transformer cette annotation ?

```
//@ assert \forall integer i ; P(i);
//@ assert \exists integer i ; P(i);
```

On ne peut pas

- On ne peut pas traduire toute la logique ACSL,
- nécessité de borner les quantifications

Outillage : le greffon E-ACSL de Frama-C

Un langage et un greffon de Frama-C

Langage :

- sous-ensemble exécutable d'ACSL
- les quantifications doivent être bornées
- toutes les constructions logiques ne sont pas supportées

Greffon :

- transforme les annotations E-ACSL en code exécutable
- de manière correcte
- modélise la mémoire pour détecter les violations

```c
int main(void){
  int a = 42 ;
  //@ assert a == 42;
}
```

On lance la commande :

```
frama-c e-acsl-t1.c -e-acsl -then-last -ocode res.c -print
```

Voir en particulier :

```c
__gen_e_acsl_assert_data.kind = "Assertion";
__gen_e_acsl_assert_data.pred_txt = "a \342\211\241 42";
__gen_e_acsl_assert_data.file = "e-acsl-t1.c";
__gen_e_acsl_assert_data.fct = "main";
__gen_e_acsl_assert_data.line = 3;
__e_acsl_assert(a == 42,& __gen_e_acsl_assert_data);
```

Compilation

`e-acsl-gcc.sh`

- permet la compilation et le *link* aux bibliothèques E-ACSL
- peut aussi lancer Frama-C et faire l'instrumentation

```
e-acsl-gcc.sh main.c -c -O output
```

Génère :

- `a.out.frama.c` le fichier instrumenté
- `output` l'exécutable sans le monitoring à l'exécution
- `output.e-acsl` l'exécutable avec le monitoring à l'exécution

Échec d'assertion

Introduisons une erreur :

```
int main(void){
  int a = 41 ;
  //@ assert a == 42;
}
```

- le fichier sans instrumentation s'exécute sans message
- avec instrumentation, une erreur est levée

```
e-acsl-t2.c: In function 'main'
e-acsl-t2.c:3: Error: Assertion failed:
  The failing predicate is:
  a == 42.
  With values:
  - a: 41
```

Exemple avec une boucle

```
int main(void){
  int a[10];
  for(int i = 0; i <= 10; i++) a[i] = 1 ;

  /*@ assert \forall integer i ;
              0 <= i < 10 ==> a[i] == 1; */
}
```

Exemple avec une boucle

```
int main(void){
  int a[10];
  for(int i = 0; i <= 10; i++) a[i] = 1 ;

  /*@ assert \forall integer i ;
              0 <= i < 10 ==> a[i] == 1; */
}
```

Remarques

- on trouve bien une boucle,
- on y trouve les contrôles à runtime,
- les potentielles RTEs du code d'origine ne sont pas instrumentées
- option `--rte=all`

Toutes les erreurs ne se valent pas

On peut utiliser E-ACSL pour robustifier un outil.
Exemple :

- monitorer avec E-ACSL mais :
  - n'échouer sévèrement que sur les *undefined behavior*
  - logger le reste

E-ACSL permet de changer l'action à réaliser sur l'assertion.

Personnaliser l'assertion

```
typedef struct eacsl_assert_data_t {
  int blocking;
  const char *kind;
  const char *name; const char *pred_txt;
  const char *file; int line; const char *fct;
  eacsl_assert_data_value_t *values;
} eacsl_assert_data_t;

void __e_acsl_assert(int pred, __e_acsl_assert_data_t * d) ;
```

```
e-acsl-gcc.sh -c file.c -O exe --external-assert=my_assert.c
```

Résumé

- E-ACSL transforme des annotations ACSL en code C,
  - cela nécessite de restreindre le langage à son fragment exécutable
- on l'utilise principalement pour monitorer du code,
- les annotations peuvent provenir de l'utilisateur ou être générées
- peut aller plus loin :
  - analyse de consommation mémoire, *use-after-free*, ...
- peut aider des outils de tests
  - détecte des comportements difficiles à détecter
  - permet d'exprimer directement des propriétés logiques
- comme complément à des outils de transformation de code
  - propriétés temporelles (Aoraï)
  - flot d'information (SecureFlow)
  - autre analyse "faite maison" ...

Partie V

Analyse statique correcte

Vérifier formellement l'absence de *runtime errors* ?

Interprétation abstraite ?

Sémantique opérationnelle

Une sémantique opérationnelle est définie par un système de règles indiquant comment calculer les expressions et les instructions du langage

- à grand pas : elle décrit comment calculer le résultat final
- à petit pas : elle décrit comment effectuer chaque étape de calcul.

Sémantique de traces

Une sémantique de trace décrit l'ensemble des traces d'exécution d'un programme.

Valeurs

L'ensemble *Val* des valeurs est l'ensemble des constantes entières.

Environnement d'évaluation

Un environnement (d'évaluation) ρ est une fonction partielle de *Var* dans *Val*. On note $\rho[x \mapsto z]$ l'environnement ρ' tel que, pour toute variable y :

$$\rho'(y) \triangleq \begin{cases} z & \text{si } x = y \\ \rho(y) & \text{sinon et si } y \in \mathrm{dom}(\rho). \end{cases}$$

On note Env_ϵ l'ensemble des environnements : $Env_\epsilon \triangleq Var \rightharpoonup Val$.

Jugement d'évaluation

On note $\rho \vdash e \Downarrow_\epsilon z$ (resp. $\rho \vdash i \Downarrow \rho'$) l'évaluation de l'expression e (resp. l'instruction i) dans l'environnement ρ conduisant à la valeur $z \in Val$ (resp. l'environnement ρ').

Évaluation des expressions entières

$$\frac{x \in \mathrm{dom}(\rho) \qquad \rho(x) = z}{\rho \vdash x \Downarrow_\epsilon z} \qquad \frac{}{\rho \vdash z \Downarrow_\epsilon z} \qquad \frac{\rho \vdash e \Downarrow_\epsilon z \qquad z' = -z}{\rho \vdash -e \Downarrow_\epsilon z'}$$

$$\frac{\rho \vdash e_1 \Downarrow_\epsilon z_1 \qquad \rho \vdash e_2 \Downarrow_\epsilon z_2 \qquad z' = z_1 + z_2}{\rho \vdash e_1 + e_2 \Downarrow_\epsilon z'}$$

$$\frac{\rho \vdash e_1 \Downarrow_\epsilon z_1 \qquad \rho \vdash e_2 \Downarrow_\epsilon z_2 \qquad z' = z_1 / z_2 \qquad z_2 \neq 0}{\rho \vdash e_1 / e_2 \Downarrow_\epsilon z'}$$

Évaluation des expressions booléennes

$$\frac{\rho \vdash e \Downarrow_\epsilon 0}{\rho \vdash !e \Downarrow_\epsilon 1} \qquad \frac{\rho \vdash e \Downarrow_\epsilon z \qquad z \,!= 0}{\rho \vdash !e \Downarrow_\epsilon 0} \qquad \frac{\rho \vdash e_1 \Downarrow_\epsilon z_1 \qquad \rho \vdash e_2 \Downarrow_\epsilon z_2 \qquad z_1 = z_2}{\rho \vdash e_1 = e_2 \Downarrow_\epsilon 1}$$

$$\frac{\rho \vdash e_1 \Downarrow_\epsilon z_1 \qquad \rho \vdash e_2 \Downarrow_\epsilon z_2 \qquad z_1 \,!= z_2}{\rho \vdash e_1 = e_2 \Downarrow_\epsilon 0} \qquad \frac{\rho \vdash e_1 \Downarrow_\epsilon z_1 \qquad \rho \vdash e_2 \Downarrow_\epsilon z_2 \qquad z_1 \leq z_2}{\rho \vdash e_1 <= e_2 \Downarrow_\epsilon 1}$$

$$\frac{\rho \vdash e_1 \Downarrow_\epsilon z_1 \qquad \rho \vdash e_2 \Downarrow_\epsilon z_2 \qquad z_1 > z_2}{\rho \vdash e_1 <= e_2 \Downarrow_\epsilon 0}$$

Évaluation des instructions

$$\frac{}{\rho \vdash \texttt{Skip} \Downarrow \rho} \qquad \frac{\rho \vdash e \Downarrow_\epsilon z}{\rho \vdash x := e \Downarrow \rho[x \mapsto z]} \qquad \frac{\rho \vdash e \Downarrow_\epsilon 0 \qquad i_2 \Downarrow \rho_2}{\rho \vdash \texttt{if } e \texttt{ then } i_1 \texttt{ else } i_2 \texttt{ fi} \Downarrow \rho_2}$$

$$\frac{\rho \vdash e \Downarrow_\epsilon z \qquad z \,!\!= 0 \qquad i_1 \Downarrow \rho_1}{\rho \vdash \texttt{if } e \texttt{ then } i_1 \texttt{ else } i_2 \texttt{ fi} \Downarrow \rho_1} \qquad \frac{\rho \vdash i_1 \Downarrow \rho_1 \qquad \rho_1 \vdash i_2 \Downarrow \rho_2}{\rho \vdash i_1 ; i_2 \Downarrow \rho_2}$$

$$\frac{\rho \vdash \texttt{if } e \texttt{ then } i ; \texttt{while } e \texttt{ do } i \texttt{ end else Skip fi} \Downarrow \rho'}{\rho \vdash \texttt{while } e \texttt{ do } i \texttt{ end} \Downarrow \rho'}$$

Exemple

$$\dfrac{\dfrac{}{\emptyset \vdash 2 \Downarrow_\epsilon 2}}{\emptyset \vdash x := 2 \Downarrow \rho_1} \qquad \dfrac{\dfrac{x \in \mathrm{dom}(\rho_1)}{\rho_1(x) = 2}}{\rho_1 \vdash x \Downarrow_\epsilon 2} \quad \dfrac{}{\rho_1 \vdash 3 \Downarrow_\epsilon 3} \quad 6 = 2 \times 3}{\rho_1 \vdash x * 3 \Downarrow_\epsilon \rho_2}}{\rho_1 \vdash x := x * 3 \Downarrow \rho_2}$$

$$\emptyset \vdash x := 2; x := x * 3 \Downarrow \rho_2$$

Notations :

- $\rho_1 = \{x \mapsto 2\}$
- $\rho_2 = \{x \mapsto 6\}$

Remarques

Déterminisme

Cette sémantique est déterministe : au plus une règle est applicable à chaque fois.

Programmes erronés et ne terminant pas

- Un programme contenant une erreur (e.g. division par zéro) n'a pas d'état résultant
- Un programme ne terminant pas non plus
- Ils sont indistinguables

Avantages et inconvénients

- ✓ proche d'une implémentation déclarative (OCaml, Prolog)
- ✗ erreurs et non-terminaison ne sont pas distinguables

Graphe de contrôle

À un programme P, on associe un graphe de contrôle étiqueté $(\mathcal{S}, \mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{F})$, où :

- À chaque instruction de P on associe un nœud de $\mathcal{S}$.
- $\mathcal{T} \subset (\mathcal{S} \times \mathcal{S})$ est l'ensemble des arcs
- $\mathcal{E} : \mathcal{S} \mapsto \{x := e \,|?e \mid \texttt{skip}\}$ associe une étiquette à chaque arc.
- $\mathcal{I} \in \mathcal{S}$ représente l'état initial
- $\mathcal{F} \in \mathcal{S}$ représente l'état final

Graphe de contrôle

À un programme P, on associe un graphe de contrôle étiqueté $(\mathcal{S}, \mathcal{T}, \mathcal{E}, \mathcal{I}, \mathcal{F})$, où :

- À chaque instruction de P on associe deux nœuds de $\mathcal{S}$ qui représentent les états avant et après l'instruction.
- $\mathcal{T} \subset (\mathcal{S} \times \mathcal{S})$ est l'ensemble des arcs
- $\mathcal{E} : \mathcal{S} \mapsto \{x := e \mid ?e \mid \texttt{skip}\}$ associe une étiquette à chaque arc.
- $\mathcal{I} \in \mathcal{S}$ représente l'état initial
- $\mathcal{F} \in \mathcal{S}$ représente l'état final
- $\epsilon \in \mathcal{S}$ représente l'état d'erreur

Graphe de contrôle - Exemple

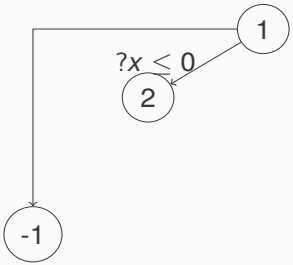
```
if(x <= 0){              // 1
  x = 1                  // 2
} else {
  ;                      // 3
}
y = 1;                   // 4
while (! (x <= 0)) {     // 5
  y = y * x;             // 6
  x--;                   // 7
}
```

Graphe de contrôle - Exemple

```
if(x <= 0){              // 1
  x = 1                  // 2
} else {
  ;                      // 3
}
y = 1;                   // 4
while (! (x <= 0)) {     // 5
  y = y * x;             // 6
  x--;                   // 7
}
```

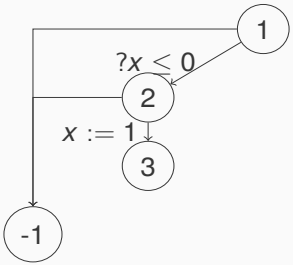
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



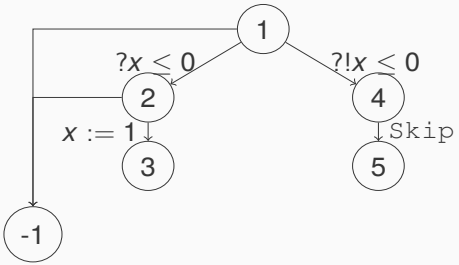
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



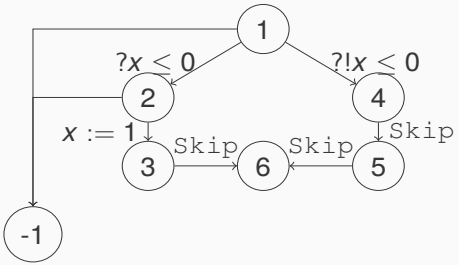
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



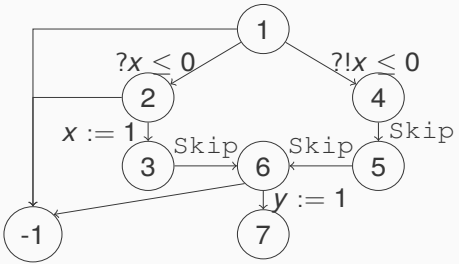
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



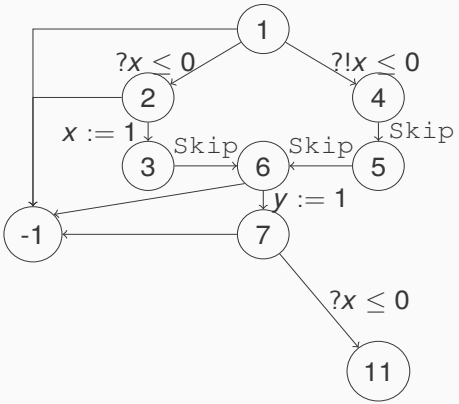
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



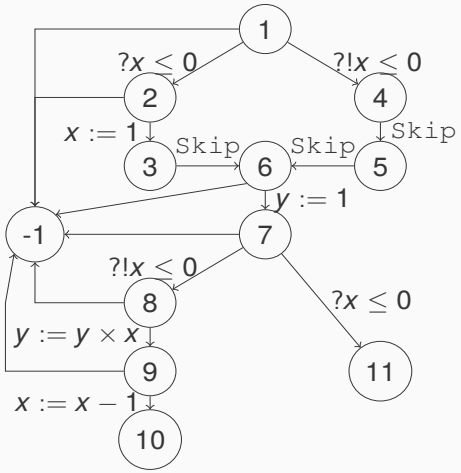
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



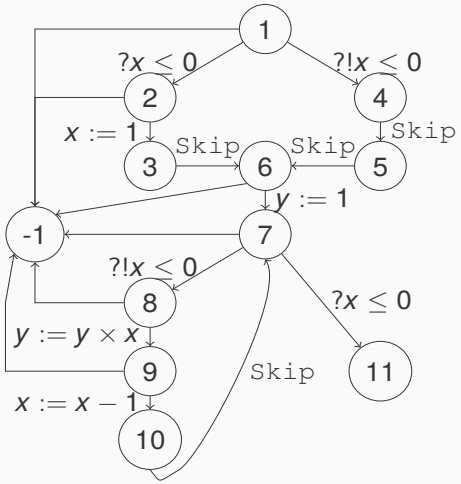
Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



Graphe de contrôle - Exemple

```
if(x <= 0){                  // 1
  x = 1                      // 2
} else {
  ;                          // 3
}
y = 1;                       // 4
while (! (x <= 0)) {         // 5
  y = y * x;                 // 6
  x--;                       // 7
}
```



Environnements et étiquettes

Skip

$$\frac{}{\rho^\flat \to^\flat \texttt{Skip} \rho^\flat}$$

Affectation

$$\frac{\rho^\flat \vdash e \Downarrow_\epsilon v}{\rho^\flat \to^\flat x := e \rho^\flat[x \mapsto v]}$$

Condition

$$\frac{\rho^\flat \vdash e \Downarrow_\epsilon v \qquad v \neq 0}{\rho^\flat \to^\flat ?e \rho^\flat}$$

Erreur

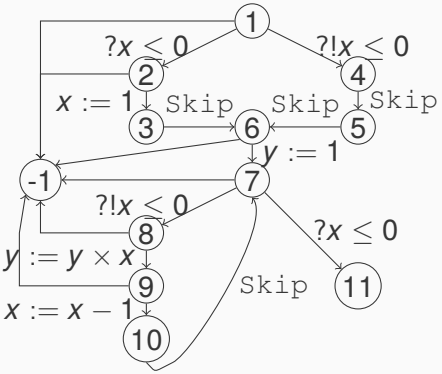
$$\frac{\forall v, \neg \rho^\flat \vdash e \Downarrow_\epsilon v}{\rho^\flat \to^\flat \mathit{Err}(e) \rho^\flat}$$

Définition

Un état concret du programme P est un couple $(s, \rho^\flat) \in \mathcal{S} \times Env_\epsilon$.
L'ensemble des transitions de P est une relation sur les états, $\Rightarrow^\flat_P$ défini par

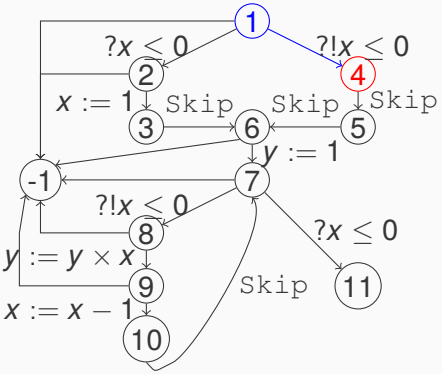
$$(s_1, \rho_1) \Rightarrow^\flat_P (s_2, \rho_2) \Leftrightarrow (s_1, s_2) \in \mathcal{T} \wedge \rho_1 \rightarrow^\flat_{\epsilon(s_1,s_2)} \rho_2$$

Exemple de trace



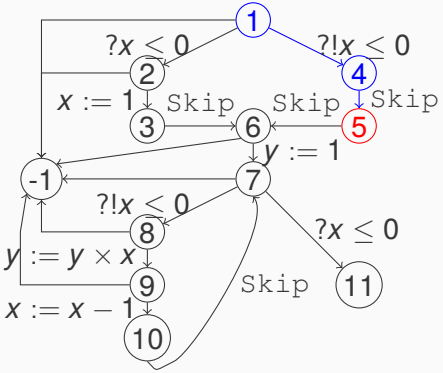
Exemple de trace

$$(s_1, \{x \mapsto 1\})$$



Exemple de trace

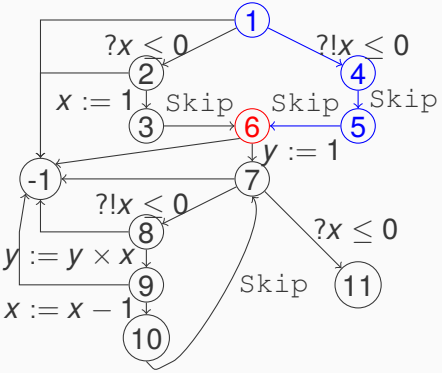
$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$



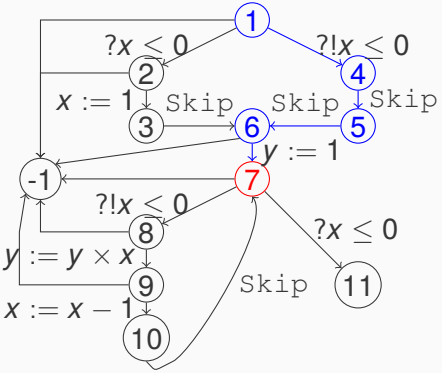
Exemple de trace

$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$



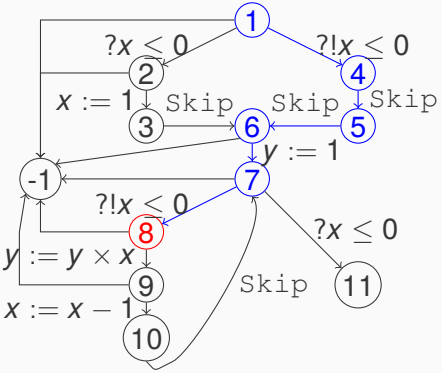
Exemple de trace



$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

Exemple de trace

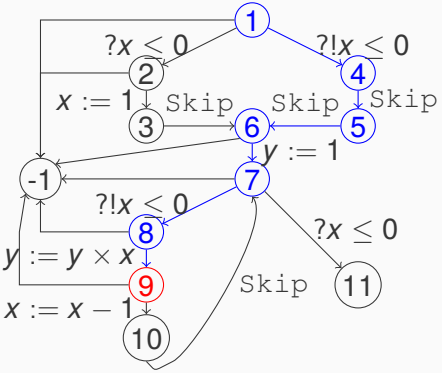


$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

$(s_7, \{x \mapsto 1, y \mapsto 1\})$

Exemple de trace



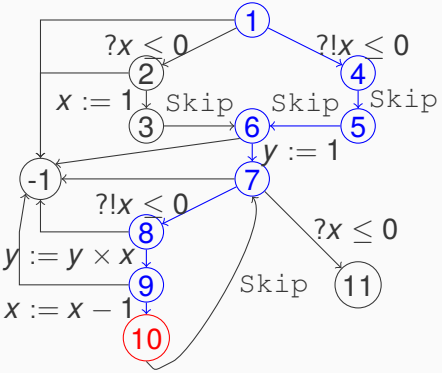
$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

$(s_7, \{x \mapsto 1, y \mapsto 1\})$

$(s_8, \{x \mapsto 1, y \mapsto 1\})$

Exemple de trace



$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

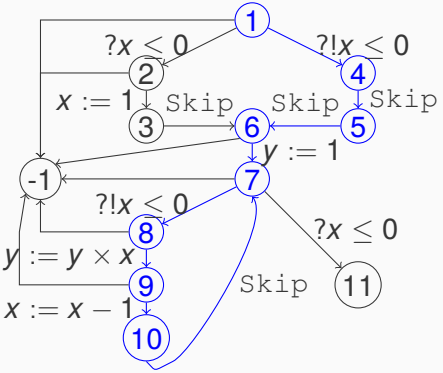
$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

$(s_7, \{x \mapsto 1, y \mapsto 1\})$

$(s_8, \{x \mapsto 1, y \mapsto 1\})$

$(s_9, \{x \mapsto 1, y \mapsto 1\})$

Exemple de trace



$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

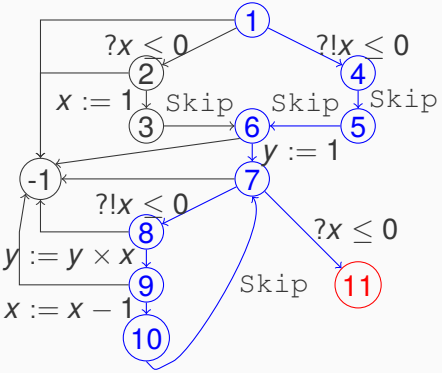
$(s_7, \{x \mapsto 1, y \mapsto 1\})$

$(s_8, \{x \mapsto 1, y \mapsto 1\})$

$(s_9, \{x \mapsto 1, y \mapsto 1\})$

$(s_{10}, \{x \mapsto 0, y \mapsto 1\})$

Exemple de trace



$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

$(s_7, \{x \mapsto 1, y \mapsto 1\})$

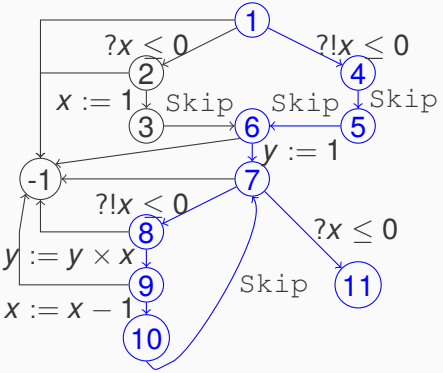
$(s_8, \{x \mapsto 1, y \mapsto 1\})$

$(s_9, \{x \mapsto 1, y \mapsto 1\})$

$(s_{10}, \{x \mapsto 0, y \mapsto 1\})$

$(s_7, \{x \mapsto 0, y \mapsto 1\})$

Exemple de trace



$(s_1, \{x \mapsto 1\})$ $(s_4, \{x \mapsto 1\})$

$(s_5, \{x \mapsto 1\})$ $(s_6, \{x \mapsto 1\})$

$(s_7, \{x \mapsto 1, y \mapsto 1\})$

$(s_8, \{x \mapsto 1, y \mapsto 1\})$

$(s_9, \{x \mapsto 1, y \mapsto 1\})$

$(s_{10}, \{x \mapsto 0, y \mapsto 1\})$

$(s_7, \{x \mapsto 0, y \mapsto 1\})$

$(s_{11}, \{x \mapsto 0, y \mapsto 1\})$

Théorème d'équivalence

Soit $(s_0, \rho_0^\flat), \cdots, (s_n, \rho_n^\flat)$ une trace finie de P.

- Si $s_n = \mathcal{F}$, $\rho_0^\flat \vdash P \Downarrow \rho_n^\flat$
- Si $s_n = \epsilon$, $\forall \rho^\flat, \neg \rho_0^\flat \vdash P \Downarrow \rho^\flat$

Réciproquement :

- Si $\rho_0^\flat \vdash P \Downarrow \rho_n^\flat$, alors il existe une trace $(s_0, \rho^\flat), \cdots, (s_n, \rho'^\flat)$
- Sinon (i.e. pas de dérivation)
  - soit il existe une trace $(s_0, \rho^\flat), \cdots, (\epsilon, \rho'^\flat)$,
  - soit il existe une trace infinie $(s_0, \rho^\flat), \cdots$

Problèmes

- L'ensemble des traces est infini
- Existence de traces infinies
- Approximation de la sémantique de traces se concentrant sur des propriétés particulières
  - Analyse flot de données
  - Correction ?
  - Terminaison ?

Bibliographie sur la sémantique

Fondations

- Gordon D. Plotkin A Structural Approach to Operational Semantics
  Tech. Rep. DAIMI FN-19, Computer Science Department, Aarhus University. 1981.
- Gilles Kahn Natural Semantics
  4th Annual Symposium on Theoretical Aspects of Computer Science.
  Springer-Verlag. London. 1987.

Ouvrages de référence :

- Glynn Winskel The formal semantics of programming language : an introduction
  MIT Press 1993.
- Carl A. Gunter Semantics of Programming Language : Structures and Techniques
  Foundations of Computing. MIT Press 1992.

Ordre partiel

Un ordre partiel $\leq$ sur un ensemble X est une relation binaire

- réflexive : $\forall x \in X, x \leq x$
- transitive : $\forall x, y, z \in X, (x \leq y \wedge y \leq z) \implies x \leq z$
- antisymétrique : $\forall x, y \in X, (x \leq y \wedge y \leq x) \implies x = y$

Treillis (complet)

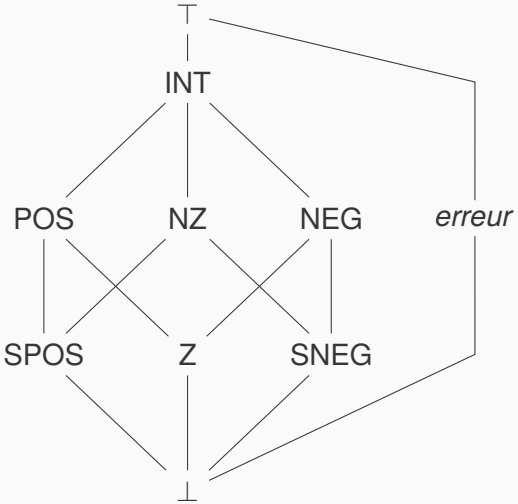
$(E, \sqsubseteq)$ est un treillis complet si et seulement si

- $\sqsubseteq$ est un ordre partiel
- $\forall F \subset E, \{x | \forall y \in F, x \sqsubseteq y\}$ possède un plus grand élément, $\sqcap F$
- $\forall F \subset E, \{x | \forall y \in F, y \sqsubseteq x\}$ possède un plus petit élément, $\sqcup F$
- $\sqcup E$ est noté $\bot$, $\sqcap E$ est noté $\top$

Chaînes

- Une chaîne est une séquence (x_i) ordonnée : $x_1 \sqsubseteq x_2 \sqsubseteq ...x_n \sqsubseteq ...$ (chaîne ascendante) ou $x_1 \sqsupseteq x_2 \sqsupseteq ...x_n \sqsupseteq ...$ (chaîne descendante)
- Un treillis complet vérifie la condition de chaîne ascendante (descendante) si et seulement si toute chaîne ascendante (descendante) est stationnaire : $\exists N, \forall i, j \geq N, x_i = x_j$

Diagramme de Hasse



Théorème du point fixe de Tarski (1953)

Toute fonction monotone f sur un treillis complet $(E, \sqsubseteq)$ admet un plus petit point fixe, $\mathrm{lfp}(f) = f(\mathrm{lfp}(f))$. De plus :

$$\mathrm{lfp}(f) = \sqcap\{x | f(x) \sqsubseteq x\}$$

Itération de Kleene

$$\begin{aligned} x_0 &= \bot \\ x_{n+1} &= f(x_n) \end{aligned}$$

- (x_i) est croissante
- Si elle converge, elle atteint $\mathrm{lfp}(f)$.
- C'est le cas si E vérifie la condition de chaîne ascendante.

Analyse flot de données (dataflow)

Vue informelle

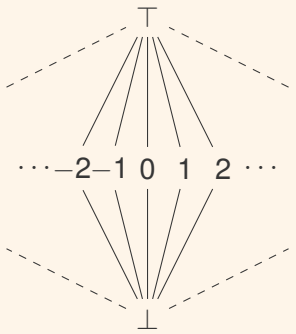
- on propage des informations le long du graphe de contrôle (fonction de transition)
- lorsqu'un sommet admet plusieurs prédécesseurs (sommet de jonction), on considère la réunion des informations propagées
- on termine lorsque la propagation des informations ne fait plus "augmenter" le résultat

Questions

- Ce procédé termine-t-il toujours ?
- Ce procédé est-il correct ?

Treillis de base

On part du treillis $\mathcal{C}$ suivant :



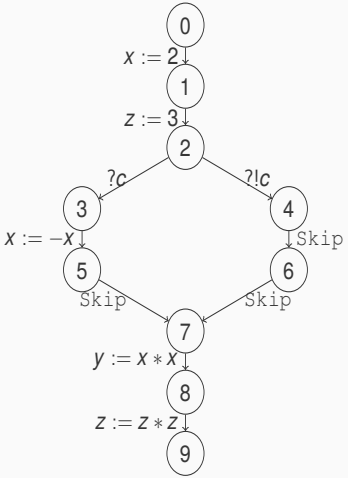
Cadre d'analyse

- $E = Var \rightharpoonup \mathcal{C}$
- $i = \lambda x.\top$
- $f_{x:=e}(\sigma^\sharp) = \sigma^\sharp[x \leftarrow [\![e]\!]_\sharp \sigma^\sharp]$
- $f_{\texttt{Skip}}(\sigma^\sharp) = \sigma^\sharp$
- $f_{?e}(\sigma^\sharp) = \begin{cases} \bot & \text{si } [\![e]\!]_\sharp \sigma^\sharp = 0 \\ \sigma^\sharp & \text{sinon} \end{cases}$

Propriétés

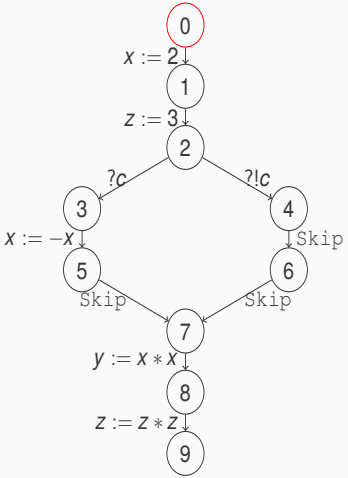
- Correction : Si à la fin de l'analyse, on a $\sigma^\sharp{}_i(x) = v$, toute trace passant par i associe v à x.
- Terminaison : L'analyse termine toujours

Exemple de propagation



nœud	c	x	y	z
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

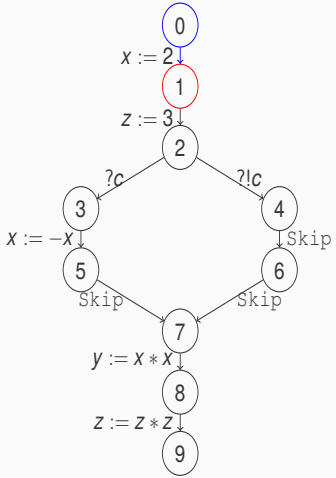
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1				
2				
3				
4				
5				
6				
7				
8				
9				

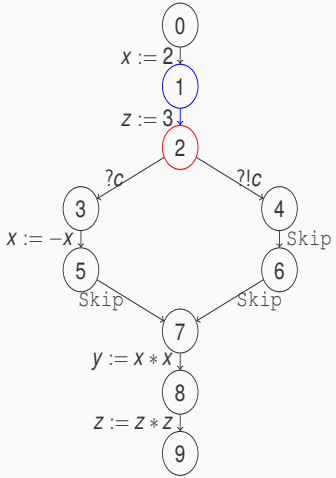
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2				
3				
4				
5				
6				
7				
8				
9				

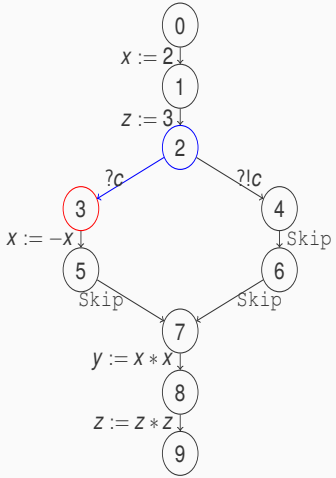
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3				
4				
5				
6				
7				
8				
9				

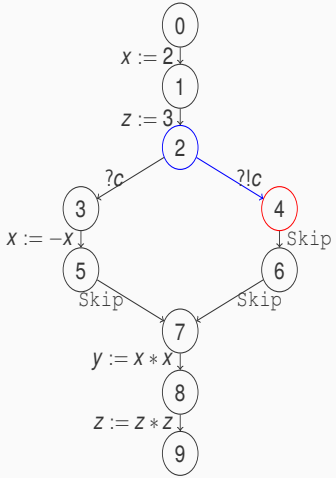
Exemple de propagation



⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4				
5				
6				
7				
8				
9				

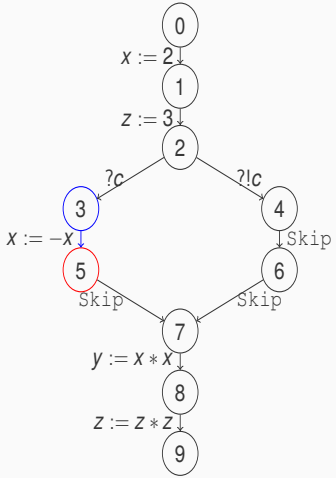
Exemple de propagation



⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4	0	2	⊤	3
5				
6				
7				
8				
9				

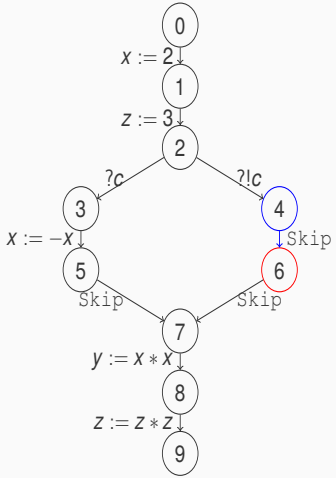
Exemple de propagation



⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4	0	2	⊤	3
5	⊤	−2	⊤	3
6				
7				
8				
9				

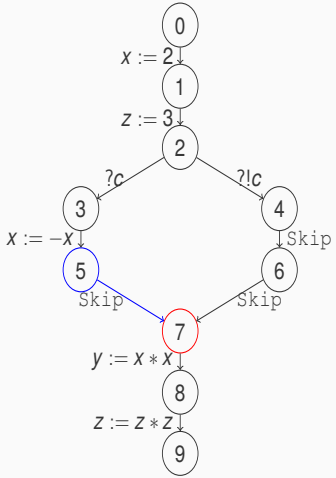
Exemple de propagation



⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4	0	2	⊤	3
5	⊤	−2	⊤	3
6	0	2	⊤	3
7				
8				
9				

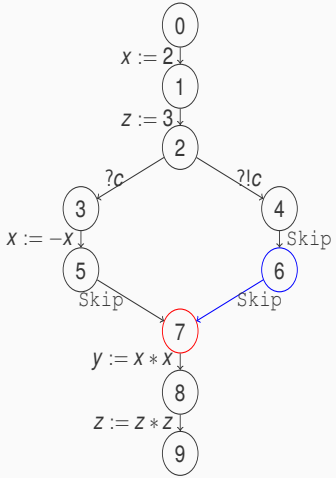
Exemple de propagation



⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4	0	2	⊤	3
5	⊤	−2	⊤	3
6	0	2	⊤	3
7	⊤	−2	⊤	3
8				
9				

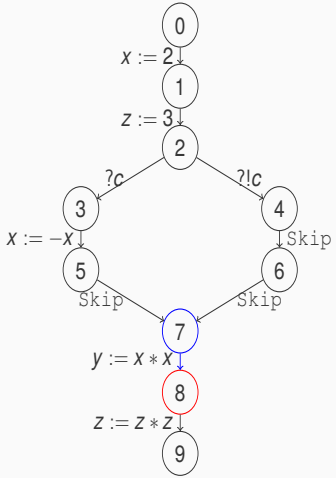
Exemple de propagation



⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4	0	2	⊤	3
5	⊤	−2	⊤	3
6	0	2	⊤	3
7	⊤	⊤	⊤	3
8				
9				

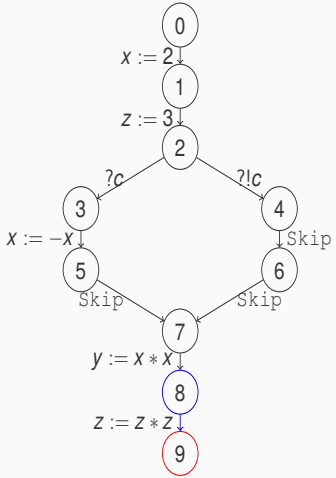
Exemple de propagation



$\top$ = "pas d'information"

nœud	c	x	y	z
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	2	$\top$	$\top$
2	$\top$	2	$\top$	3
3	$\top$	2	$\top$	3
4	0	2	$\top$	3
5	$\top$	-2	$\top$	3
6	0	2	$\top$	3
7	$\top$	$\top$	$\top$	3
8	$\top$	$\top$	$\top$	3
9				

Exemple de propagation



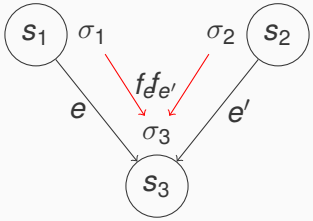
⊤ = "pas d'information"

nœud	c	x	y	z
0	⊤	⊤	⊤	⊤
1	⊤	2	⊤	⊤
2	⊤	2	⊤	3
3	⊤	2	⊤	3
4	0	2	⊤	3
5	⊤	−2	⊤	3
6	0	2	⊤	3
7	⊤	⊤	⊤	3
8	⊤	⊤	⊤	3
9	⊤	⊤	⊤	9

Sous- et Sur- approximation

Lorsqu'un sommet a plusieurs prédécesseurs (ou successeurs pour une analyse arrière), il faut intégrer les informations qui en proviennent :

- Sous-approximation : $\sigma(s) = \bigsqcap_{s' \in \mathrm{Pred}(s)} \{f_{\epsilon(s',s)}(\sigma(s'))\}$ (on prend l'information qui est vraie sur tous les chemins).
- Sur-approximation : $\sigma(s) = \bigsqcup_{s' \in \mathrm{Pred}(s)} \{f_{\epsilon(s',s)}(\sigma(s'))\}$ (on prend l'information qui est vraie sur au moins un chemin).



Définition

À chaque point de programme s d'un programme P on associe l'ensemble :

$$\mathcal{C}_P(s) = \left\{ \begin{array}{ll} \rho^\flat \in Env^\flat \mid & \exists (s_0, \rho_0), (s_1, \rho_1), \cdots, (s_i, \rho_i), \cdots \in \mathbb{T}(P) \\ & \exists k \ s = s_k \text{ et } \rho^\flat = \rho_k \end{array} \right\}$$

$$\mathcal{C}_P \in Env_P^\flat = \{f^\flat : \mathcal{S} \to \wp(Env^\flat)\}$$

Note

$Env_P^\flat$ peut être muni d'un ordre partiel :

$$\forall f_1^\flat, f_2^\flat \in Env_c^\flat \qquad f_1^\flat \subseteq f_2^\flat \quad \text{ssi} \quad \forall s \in \mathcal{S},\ f_1^\flat(s) \subseteq f_2^\flat(s)$$

Cadre d'analyse

On cherche à calculer la sémantique collectrice de P, $Coll_P$, comme une analyse flot de données.

- On se donne un ensemble d'états initiaux arbitraires $R_{\mathcal{I}}$
- Les fonctions de transition sont de la forme :

$$\mathcal{P}^\flat_e = \lambda R.\{\rho \mid \exists \rho' \in R, \rho' \rightarrow^\flat_e \rho\}$$

- On part de $\lambda s \in \mathcal{S}. \begin{cases} R_{\mathcal{I}} & \text{si } s = \mathcal{I} \\ \emptyset & \text{sinon} \end{cases}$ (qui appartient à $Env^\flat_P$)

On cherche à calculer la sémantique collectrice de P, $Coll_P$, comme une analyse flot de données.

- On se donne un ensemble d'états initiaux arbitraires $R_{\mathcal{I}}$
- Les fonctions de transition sont de la forme :

$$\mathcal{P}_e^\flat = \lambda R.\{\rho \mid \exists \rho' \in R, \rho' \rightarrow^\flat_e \rho\}$$

- On part de $\lambda s \in \mathcal{S}.\begin{cases} R_{\mathcal{I}} & \text{si } s = \mathcal{I} \\ \emptyset & \text{sinon} \end{cases}$ (qui appartient à $Env_P^\flat$)

Lemme

$\mathcal{P}_e^\flat$ est une fonction monotone

On cherche à calculer la sémantique collectrice de P, $Coll_P$, comme une analyse flot de données.

- On se donne un ensemble d'états initiaux arbitraires $R_{\mathcal{I}}$
- Les fonctions de transition sont de la forme :

$$\mathcal{P}^\flat_e = \lambda R.\{\rho \mid \exists \rho' \in R, \rho' \rightarrow^\flat_e \rho\}$$

- On part de $\lambda s \in \mathcal{S}.\begin{cases} R_{\mathcal{I}} & \text{si } s = \mathcal{I} \\ \emptyset & \text{sinon} \end{cases}$ (qui appartient à $Env^\flat_P$)

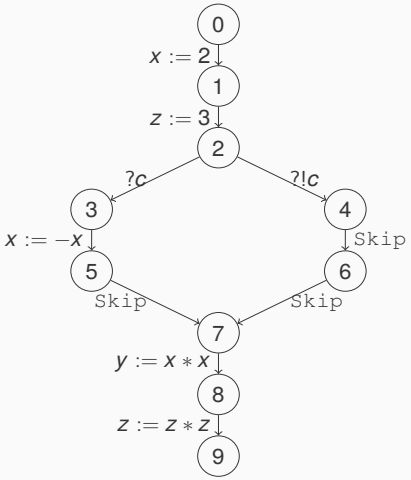
Lemme

$\mathcal{P}^\flat_e$ est une fonction monotone

Remarque

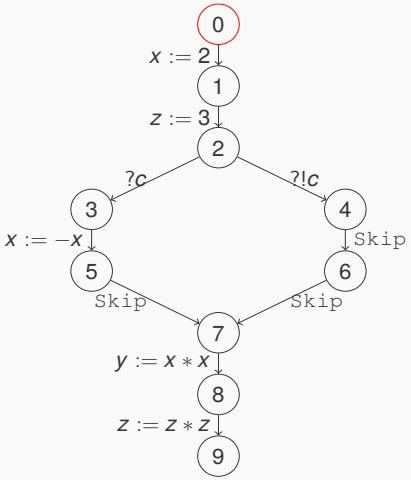
Par contre $\wp(Env^\flat_P)$ ne vérifie pas la condition de chaîne ascendante.

Exemple



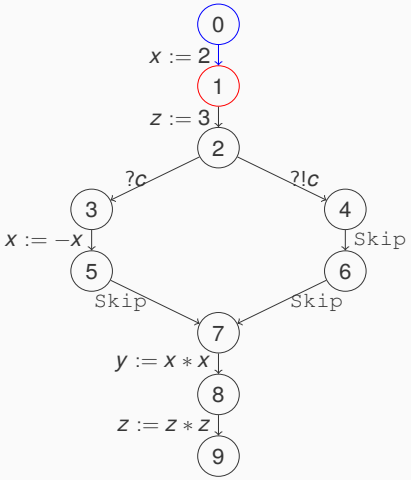
nœuds	environnements
0	
1	
2	
3	
4	
5	
6	

Exemple



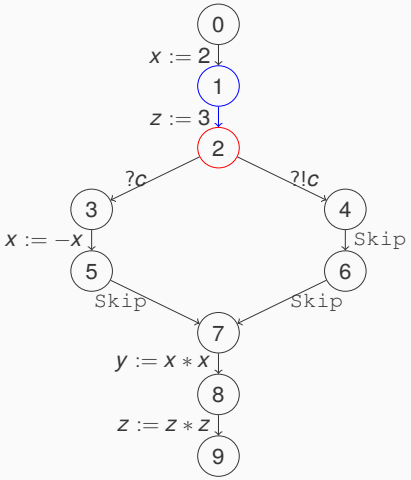
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	
2	
3	
4	
5	
6	

Exemple



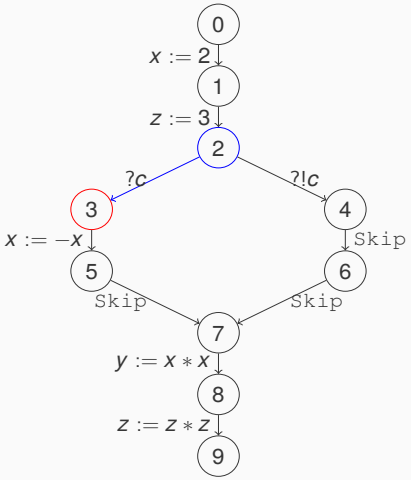
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow n_z; c \leftarrow n_c\}$
2	
3	
4	
5	
6	

Exemple



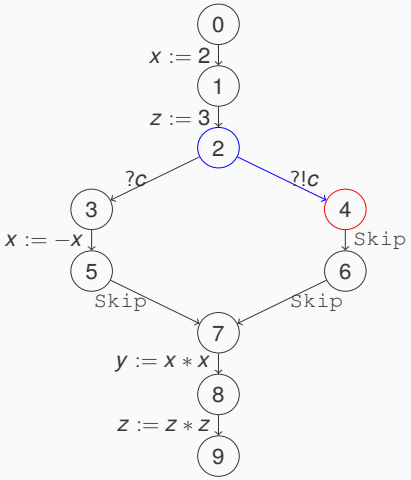
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	$\{x \leftarrow 2; y \leftarrow n_y; z \leftarrow n_z; c \leftarrow n_c\}$
2	$\{x \leftarrow 2; y \leftarrow n_y; z \leftarrow 3; c \leftarrow n_c\}$
3	
4	
5	
6	

Exemple



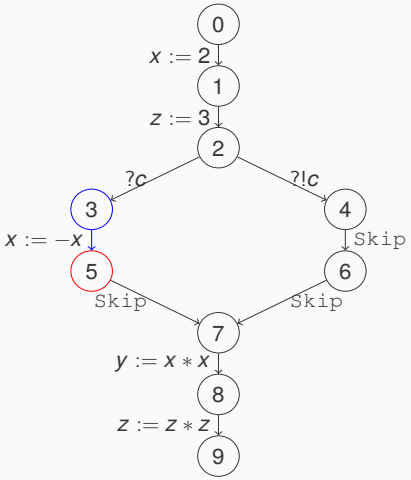
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow n_z; c \leftarrow n_c\}$
2	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}$
3	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
4	
5	
6	

Exemple



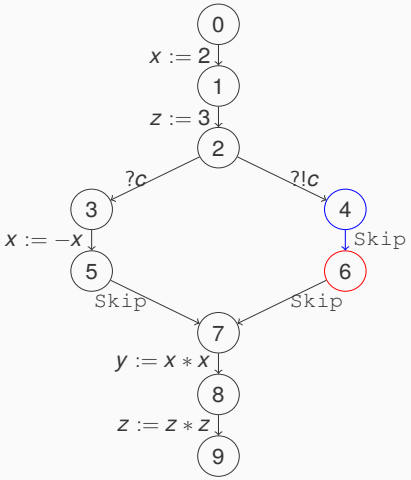
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow n_z; c \leftarrow n_c\}$
2	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}$
3	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
4	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow 0\}$
5	
6	

Exemple



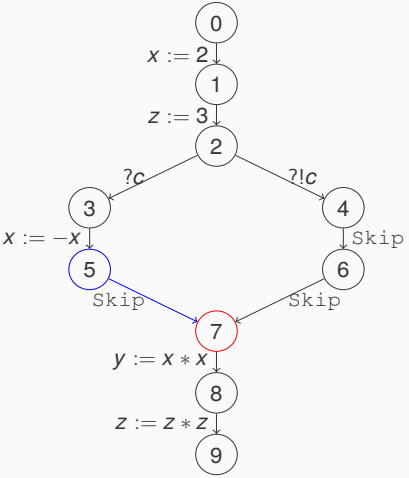
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow n_z; c \leftarrow n_c\}$
2	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}$
3	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
4	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow 0\}$
5	$\{x \leftarrow -2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
6	

Exemple



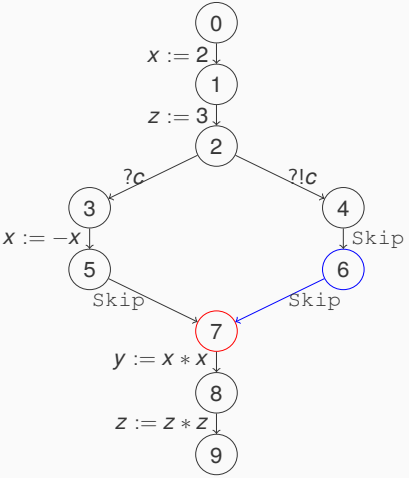
nœuds	environnements
0	$\{\lambda x.n_x \mid n_x \in \mathbb{Z}\}$
1	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow n_z; c \leftarrow n_c\}$
2	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}$
3	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
4	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow 0\}$
5	$\{x \leftarrow -2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
6	$\{x \leftarrow 2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow 0\}$

Exemple (suite)



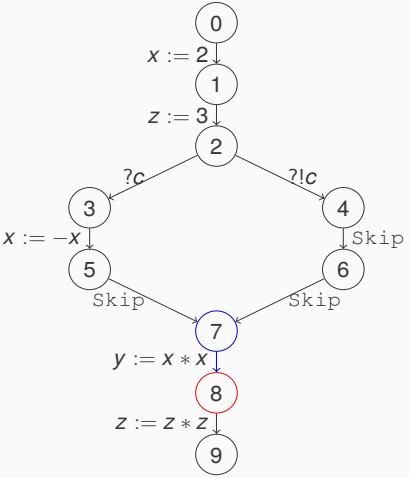
nœuds	environnements
7	$\{x \leftarrow -2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$
8	
9	

Exemple (suite)



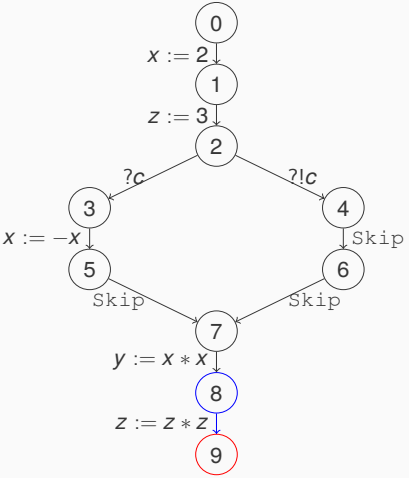
nœuds	environnements
7	$\{x \leftarrow -2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$ $\cup \{x \leftarrow 2; y \leftarrow n_y$ $z \leftarrow 3; c \leftarrow 0\}$
8	
9	

Exemple (suite)



nœuds	environnements
7	$\{x \leftarrow -2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$ $\cup\{x \leftarrow 2; y \leftarrow n_y$ $z \leftarrow 3; c \leftarrow 0\}$
8	$\{x \leftarrow -2; y \leftarrow 4;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$ $\cup\{x \leftarrow 2; y \leftarrow 4$ $z \leftarrow 3; c \leftarrow 0\}$
9	

Exemple (suite)



nœuds	environnements
7	$\{x \leftarrow -2; y \leftarrow n_y;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$ $\cup\{x \leftarrow 2; y \leftarrow n_y$ $z \leftarrow 3; c \leftarrow 0\}$
8	$\{x \leftarrow -2; y \leftarrow 4;$ $z \leftarrow 3; c \leftarrow n_c\}; n_c \neq 0$ $\cup\{x \leftarrow 2; y \leftarrow 4$ $z \leftarrow 3; c \leftarrow 0\}$
9	$\{x \leftarrow -2; y \leftarrow 4;$ $z \leftarrow 9; c \leftarrow n_c\}; n_c \neq 0$ $\cup\{x \leftarrow 2; y \leftarrow 4$ $z \leftarrow 9; c \leftarrow 0\}$

Théorème

Tout état associé à un point de programme s dans une trace de P est dans $Coll_P(s)$:

$$\mathcal{C}_P = Coll_P$$

Approximation

On perd la relation entre les états : Si s_1 et s_2 sont deux états reliés par un arc, avec $\rho_1 \in Coll_P(s_1)$, $\rho_2 \in Coll_P(s_2)$, on n'a pas forcément de trace contenant $(s_1, \rho_1), (s_2, \rho_2)$

Terminaison

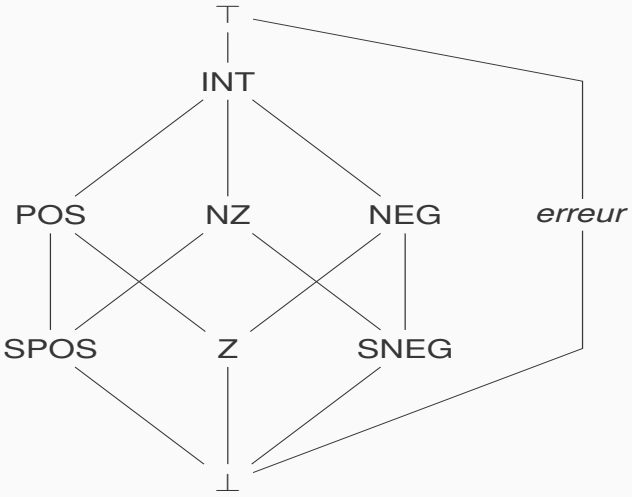
Calculer $Coll_P$ par itération ne termine pas toujours. Nous voulons des abstractions $\mathcal{A}$

- correctes : telles que $Coll_P(s) \subset \mathcal{A}(s)$
- qui terminent toujours, quel que soit P

Principe de l'interprétation abstraite

- définir un domaine abstrait
- définir les fonctions de correspondance entre domaines concret et abstrait
  - abstraction α
  - concrétisation γ
- preuve que (α, γ) est une connexion de Galois
- déduction "automatique" des autres éléments abstraits
  - environnements abstraits
  - opérateurs abstraits
  - fonction de transition abstraite
- analyse flot de données avec ces opérations abstraites
- garantie "automatique" de correction et de terminaison

Exemple : treillis des signes



Exemple : Abstraction

On va considérer des environnements abstraits, $\mathbb{E}^\sharp : \mathit{Var} \rightharpoonup$ Signes. Chacun représente un ensemble d'environnements concrets :

$$\begin{aligned}
\gamma(\top) &= \mathbb{Z} \cup \{\mathit{erreur}\} \\
\gamma(\mathsf{INT}) &= \mathbb{Z} \\
\gamma(\mathsf{POS}) &= \{v \in \mathbb{Z} \mid v \geq 0\} \\
\gamma(\mathsf{NEG}) &= \{v \in \mathbb{Z} \mid v \leq 0\} \\
\gamma(\mathsf{SPOS}) &= \{v \in \mathbb{Z} \mid v > 0\} \\
\gamma(\mathsf{SNEG}) &= \{v \in \mathbb{Z} \mid v < 0\} \\
\gamma(\mathsf{Z}) &= \{0\} \\
\gamma(\mathsf{NZ}) &= \{v \in \mathbb{Z} \mid v \neq 0\} \\
\gamma(\bot) &= \emptyset \\
\gamma(\rho^\sharp) &= \{\rho^\flat \mid \forall x, \rho^\flat(x) \in \gamma(\rho^\sharp(x))\}
\end{aligned}$$

Exemple : Opérations abstraites

Les fonctions de transitions sont assez similaires aux opérations concrètes, sauf qu'on évalue les expressions dans Signes

$+^\sharp$	$\top$	$\bot$	INT	*erreur*	POS	NEG	NZ	SPOS	Z	SNEG
$\top$										
$\bot$										
INT										
erreur										
POS										
NEG										
NZ										
SPOS										
Z										
SNEG										

Exemple : Opérations abstraites

Les fonctions de transitions sont assez similaires aux opérations concrètes, sauf qu'on évalue les expressions dans Signes

$+^\sharp$	$\top$	$\bot$	INT	*erreur*	POS	NEG	NZ	SPOS	Z	SNEG
$\top$	$\top$	$\bot$	$\top$	*erreur*	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
INT	$\top$	$\bot$	INT	*erreur*	INT	INT	INT	INT	INT	INT
erreur	*erreur*	$\bot$	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*	*erreur*
POS	$\top$	$\bot$	INT	*erreur*	POS	INT	INT	SPOS	POS	INT
NEG	$\top$	$\bot$	INT	*erreur*	INT	NEG	INT	INT	NEG	SNEG
NZ	$\top$	$\bot$	INT	*erreur*	INT	INT	INT	INT	INT	INT
SPOS	$\top$	$\bot$	INT	*erreur*	SPOS	INT	INT	SPOS	SPOS	INT
Z	$\top$	$\bot$	INT	*erreur*	POS	NEG	Int	SPOS	Z	SNEG
SNEG	$\top$	$\bot$	INT	*erreur*	INT	SNEG	INT	INT	SNEG	SNEG

Exemple : fonctions de transition abstraites

Skip

$$\frac{}{\rho^\sharp \to^\sharp_{\texttt{Skip}} \rho^\sharp}$$

Affectation

$$\frac{\rho^\sharp \vdash e \Downarrow^\sharp_\epsilon v \qquad v \notin \{\bot \mid \mathit{erreur}\}}{\rho^\sharp \to^\sharp_{x:=e} \rho^\sharp[x \mapsto v]}$$

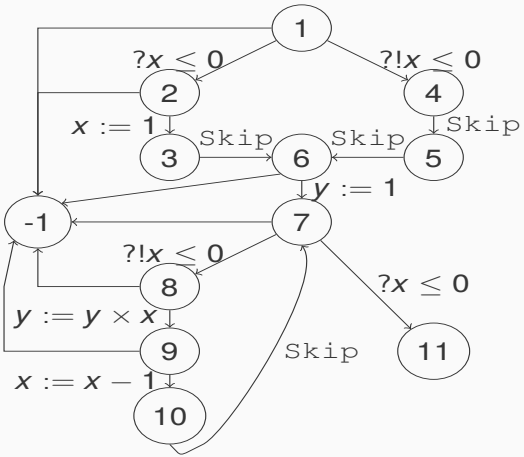
Condition

$$\frac{\rho^\sharp \vdash e \Downarrow^\sharp_\epsilon v \qquad v \notin \{\mathsf{Z} \mid \bot \mid \text{erreur}\}}{\rho^\sharp \to^\sharp_{?e} \left\lfloor \rho^\sharp \right\rfloor}$$

Erreur

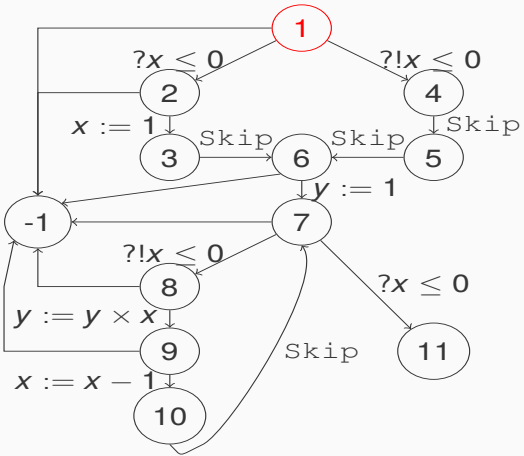
$$\frac{\rho^\sharp \vdash e \Downarrow^\sharp_\epsilon v \qquad v \in \{\top \mid \mathit{erreur}\}}{\rho^\sharp \to^\sharp_{Err(e)} \rho^\sharp}$$

Exemple : Analyse



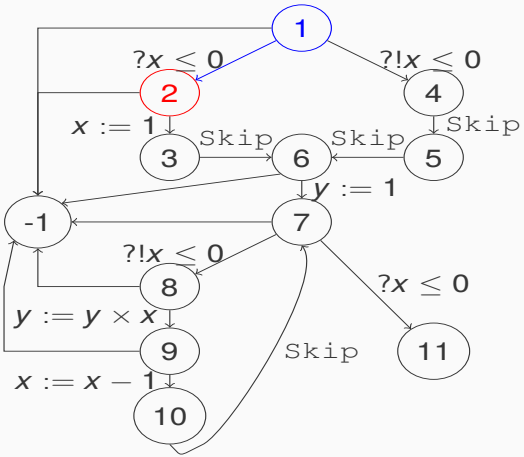
nœud	x	y
s_1		
s_2		
s_3		
s_4		
s_5		
s_6		
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



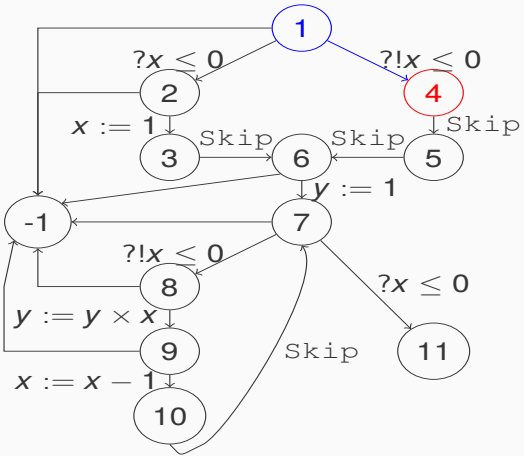
nœud	x	y
s_1	INT	INT
s_2		
s_3		
s_4		
s_5		
s_6		
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



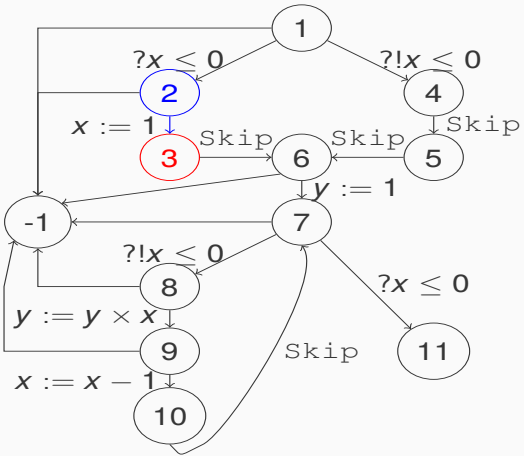
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3		
s_4		
s_5		
s_6		
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



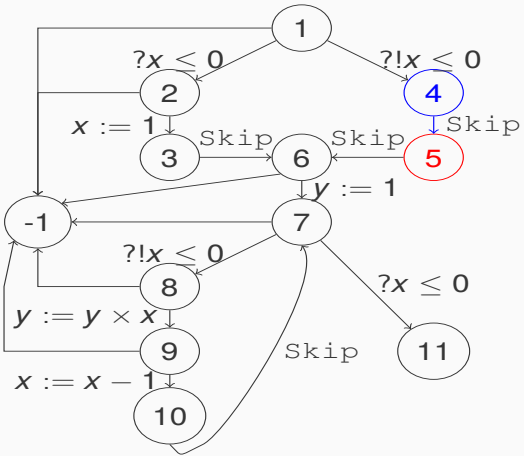
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3		
s_4	SPOS	INT
s_5		
s_6		
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



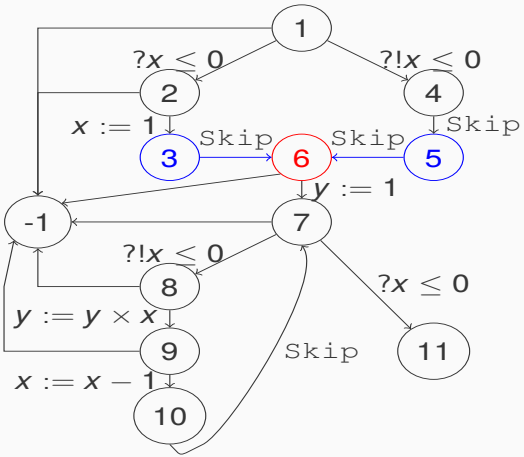
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5		
s_6		
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



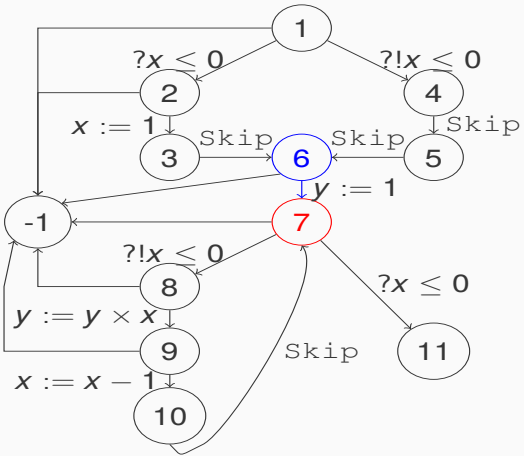
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6		
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



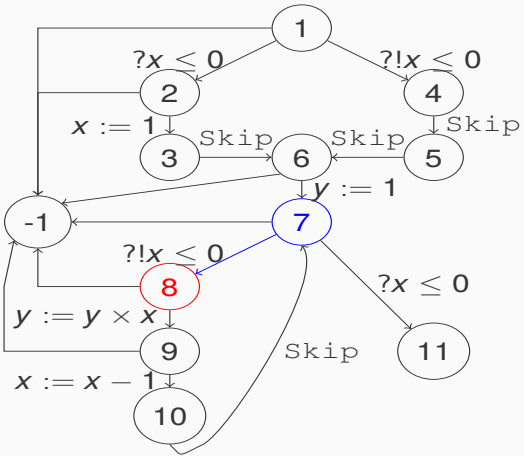
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7		
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



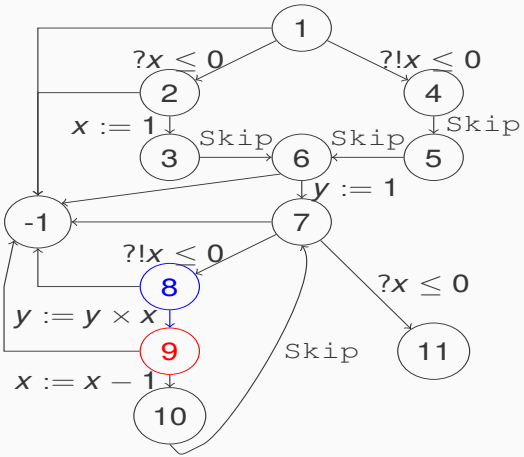
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	SPOS	SPOS
s_8		
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



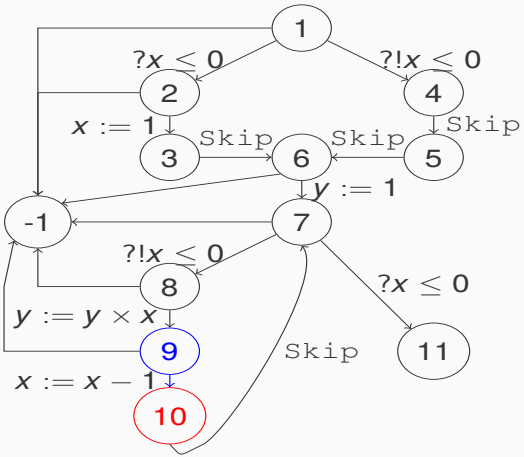
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	SPOS	SPOS
s_8	SPOS	SPOS
s_9		
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



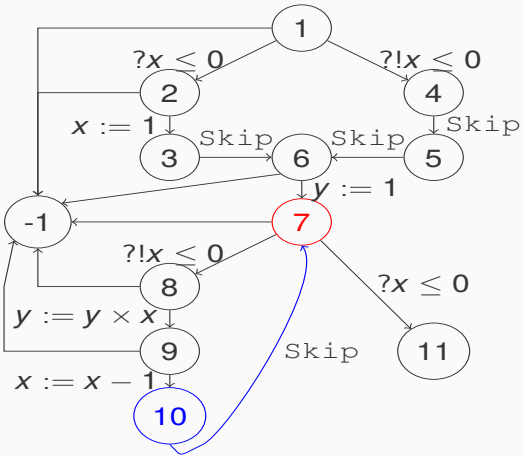
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	SPOS	SPOS
s_8	SPOS	SPOS
s_9	SPOS	SPOS
s_{10}		
s_{11}		
s_{-1}		

Exemple : Analyse



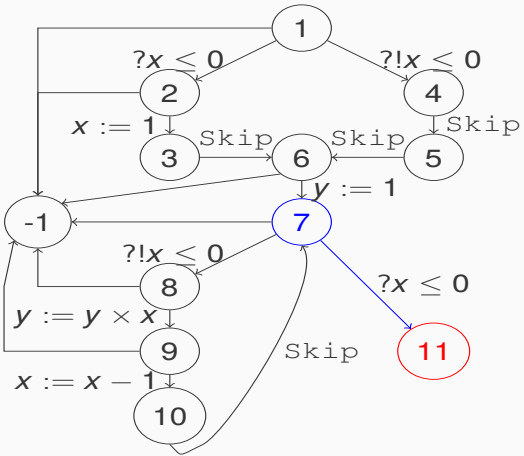
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	SPOS	SPOS
s_8	SPOS	SPOS
s_9	SPOS	SPOS
s_{10}	INT	SPOS
s_{11}		
s_{-1}		

Exemple : Analyse



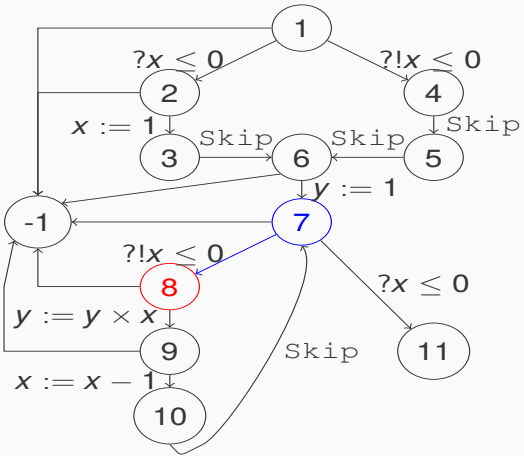
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	INT	SPOS
s_8	SPOS	SPOS
s_9	SPOS	SPOS
s_{10}	INT	SPOS
s_{11}		
s_{-1}		

Exemple : Analyse



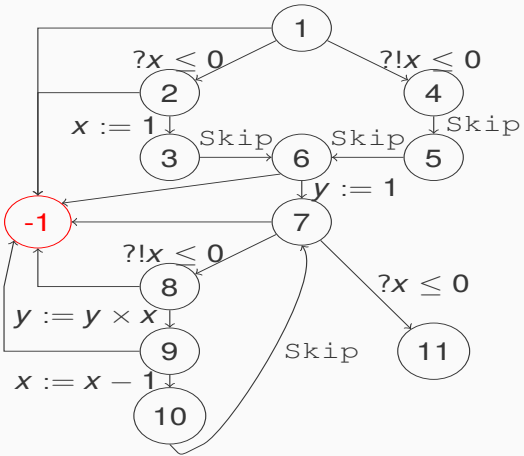
nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	INT	SPOS
s_8	SPOS	SPOS
s_9	SPOS	SPOS
s_{10}	INT	SPOS
s_{11}	NEG	SPOS
s_{-1}		

Exemple : Analyse



nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	INT	SPOS
s_8	SPOS	SPOS
s_9	SPOS	SPOS
s_{10}	INT	SPOS
s_{11}	NEG	SPOS
s_{-1}		

Exemple : Analyse



nœud	x	y
s_1	INT	INT
s_2	NEG	INT
s_3	SPOS	INT
s_4	SPOS	INT
s_5	SPOS	INT
s_6	SPOS	INT
s_7	INT	SPOS
s_8	SPOS	SPOS
s_9	SPOS	SPOS
s_{10}	INT	SPOS
s_{11}	NEG	SPOS
s_{-1}	$\bot$	$\bot$

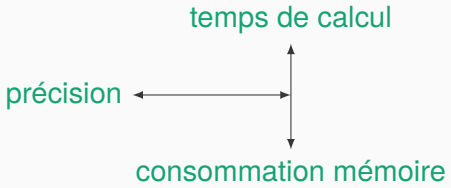
Propriétés

- [Terminaison] L'analyse des signes termine toujours.
- [Correction] Soit s un sommet du graphe de contrôle de P et x une variable. $Coll_P(s)(x) \in \gamma(\rho^\sharp(s)(x))$ où $\rho^\sharp(s)$ est l'environnement associé à s par l'analyse.
- La correction peut se montrer "à la main", en raisonnant par induction sur P, mais il existe une manière systématique de construire l'analyse qui garantit que c'est correct.

Une question de choix

En pratique, bien choisir son domaine abstrait est fondamental

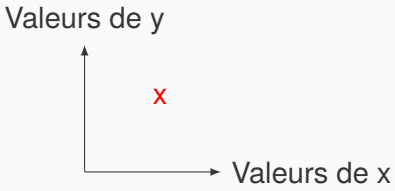
- doit être suffisamment précis
- en particulier, doit permettre d'énoncer la propriété souhaitée
- doit être calculable pour un coût temps/mémoire raisonnable



- domaines non relationnels : aucune relation conservée entre les éléments du domaine $\Rightarrow$ peu précis, mais pas cher
- domaines relationnels : relations entre éléments du domaine $\Rightarrow$ plus précis, mais plus cher

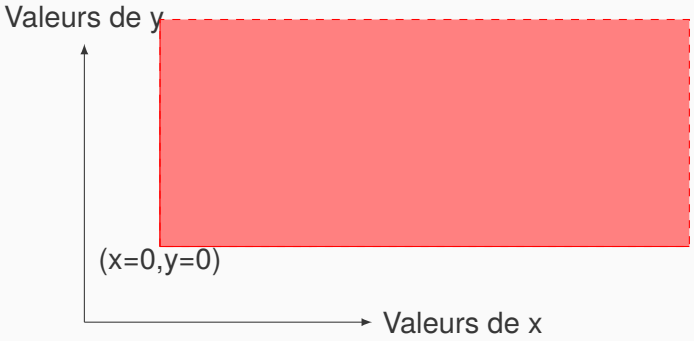
Domaines des constantes

- $x = z$ $(z \in \mathbb{Z})$
- domaine non relationnel
- si la valeur exacte n'est pas connue, perte de toute information



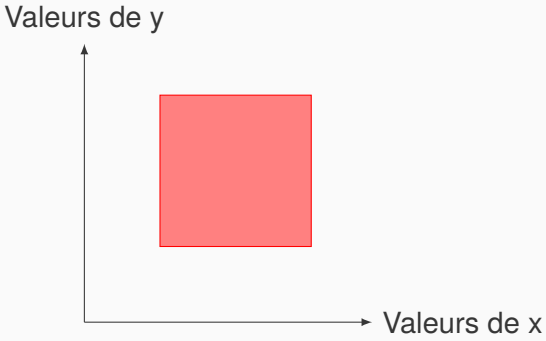
Domaine des signes

- $x \text{ op } 0$ op $\in \{\geq, >, \leq, <, =, \neq\}$
- domaine non relationnel,
- conservation de la polarité des valeurs possibles



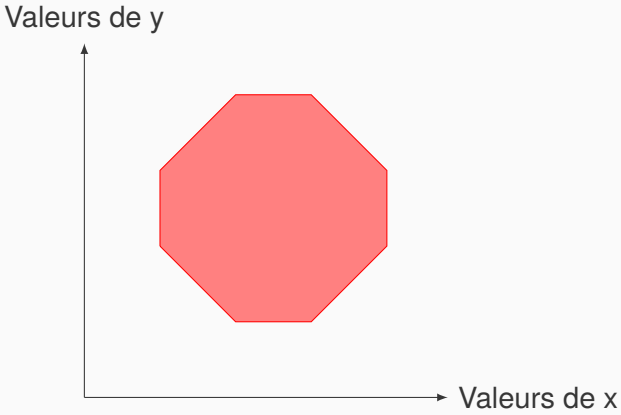
Domaine des intervalles

- $x \in [i_0, i_1]$
- domaine non relationnel,
- conservation d'un intervalle regroupant toutes les valeurs possibles



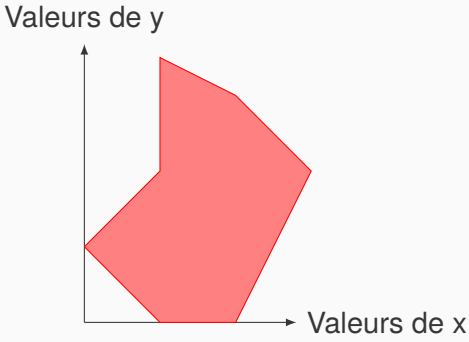
Domaine des octogones

- $\pm x \pm y \leq c$
- domaine relationnel
- conservation de relations linéaires simples entre éléments



Domaine des polyèdres

- $kx + ly \leq c$
- domaine relationnel
- relations linéaires complexes entre deux éléments

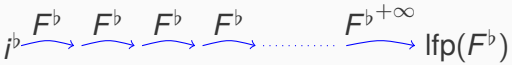


Abstraction et concrétisation

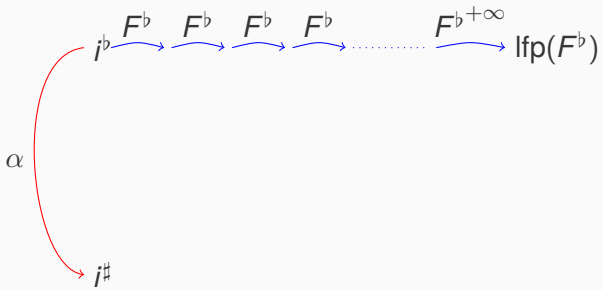
- treillis concret $(L^\flat, \sqsubseteq^\flat)$ pour l'analyse concrète
- treillis abstrait $(L^\sharp, \sqsubseteq^\sharp)$ pour l'analyse abstraite
- relation d'ordre $\sqsubseteq^\sharp$: "être moins précis que"
- opérateur d'abstraction α : transforme un environnement concret en un abstrait
- opérateur de concrétisation γ : transforme un environnement abstrait en un concret

$$(L^\flat, \sqsubseteq^\flat) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} (L^\sharp, \sqsubseteq^\sharp)$$

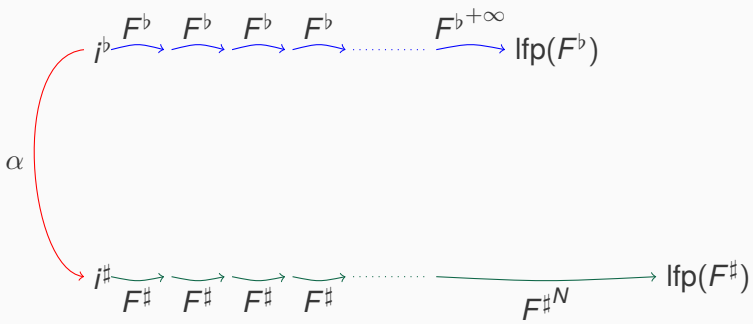
Objectif



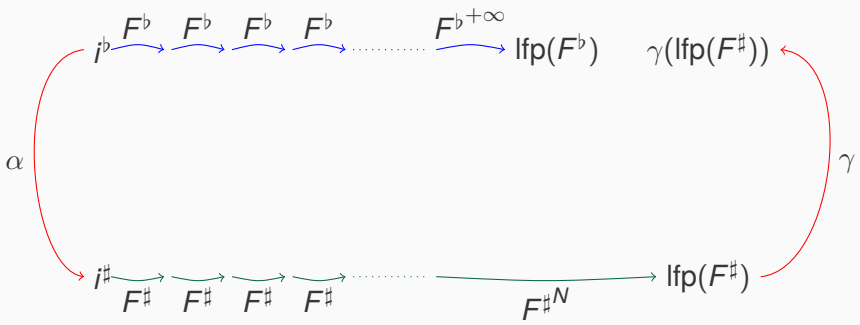
Objectif


$i^\flat$
$F^\flat$
$F^\flat$
$F^\flat$
$F^\flat$
$F^{\flat+\infty}$
$\mathrm{lfp}(F^\flat)$
α
$i^\sharp$

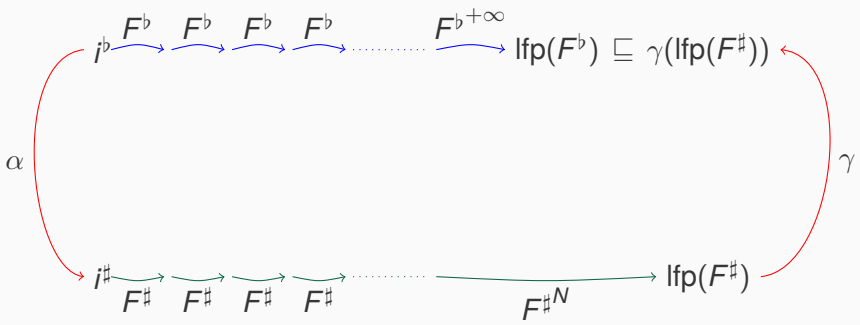
Objectif


$F^\flat$
$F^\flat$
$F^\flat$
$F^\flat$
$F^{\flat+\infty}$
$i^\flat$
$\mathrm{lfp}(F^\flat)$
α
$i^\sharp$
$F^\sharp$
$F^\sharp$
$F^\sharp$
$F^\sharp$
$F^{\sharp N}$
$\mathrm{lfp}(F^\sharp)$

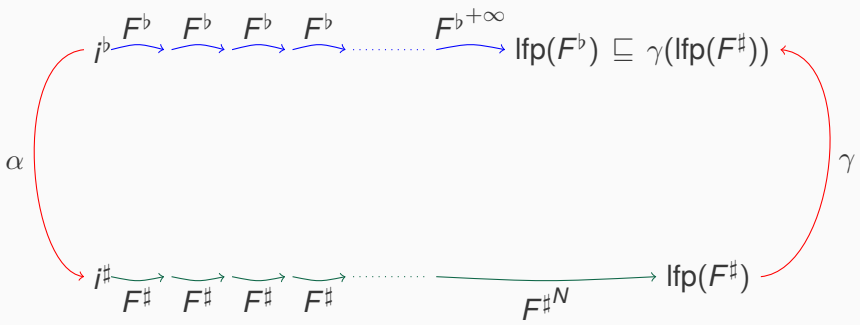
Objectif


$F^\flat$
$F^\flat$
$F^\flat$
$F^\flat$
$F^{\flat+\infty}$
$i^\flat$
$\mathrm{lfp}(F^\flat)$
$\gamma(\mathrm{lfp}(F^\sharp))$
α
γ
$i^\sharp$
$F^\sharp$
$F^\sharp$
$F^\sharp$
$F^\sharp$
$F^{\sharp N}$
$\mathrm{lfp}(F^\sharp)$

Objectif


$F^\flat$ $F^\flat$ $F^\flat$ $F^\flat$ $F^{\flat+\infty}$
$i^\flat$
$\mathrm{lfp}(F^\flat) \sqsubseteq \gamma(\mathrm{lfp}(F^\sharp))$
α
γ
$i^\sharp$
$F^\sharp$ $F^\sharp$ $F^\sharp$ $F^\sharp$
$F^{\sharp N}$
$\mathrm{lfp}(F^\sharp)$

Objectif



- Conditions sur α et γ
- Comment choisir $F^\sharp$?

Composition de α et γ

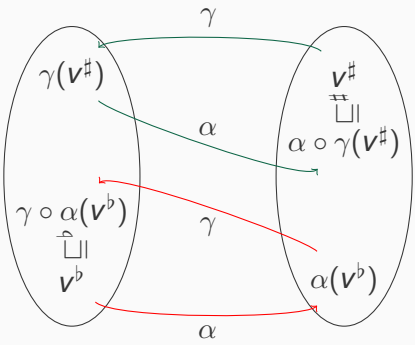
Lemme

(α, γ) est une connexion de Galois si et seulement si les trois conditions suivantes sont satisfaites :

1 ⟩ α et γ sont monotones.

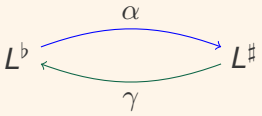
2 ⟩ $\forall v^\flat \in L^\flat$, $v^\flat \sqsubseteq^\flat (\gamma \circ \alpha)(v^\flat)$.

3 ⟩ $\forall v^\sharp \in L^\sharp$, $(\alpha \circ \gamma)(v^\sharp) \sqsubseteq^\sharp v^\sharp$



Théorème (Cousot)

Soit une connexion de Galois entre treillis complet $L^\flat$ et $L^\sharp$:



Si $f : L^\flat \to L^\flat$ est une fonction monotone alors :

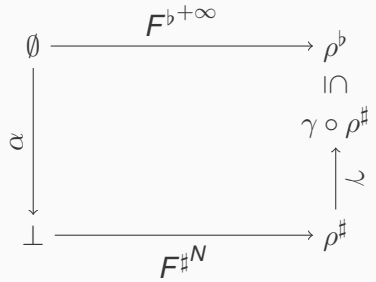
$$\mathrm{lfp}(f) \sqsubseteq^\flat \gamma(\mathrm{lfp}(\alpha \circ f \circ \gamma)).$$

Monotonie

Soit $(L, \sqsubseteq, \sqcup, \sqcap)$ un treillis complet, et f et g deux fonctions monotones de D dans D.

$$\text{si pour tout } x \in L, f(x) \sqsubseteq g(x), \text{ alors } \mathrm{lfp}(f) \sqsubseteq \mathrm{lfp}(g).$$

Point fixe abstrait



Le théorème de Cousot nous donne un bon candidat pour $F^\sharp$:

$$F^\sharp = \alpha \circ F^\flat \circ \gamma$$

- correct “automatiquement” par construction,
- mais pas forcément calculable ($F^\flat$ elle-même ne l’est pas)
- approximation plus grossière possible grâce au théorème de monotonie

Treillis des signes : Opérateur d'abstraction

$$\alpha \quad : \quad \mathcal{P}(\mathbb{Z}) \cup \{\text{erreur}\} \to \textit{Signes}$$

$$\begin{aligned}
\alpha(\{0\}) &= \mathsf{Z} \\
\alpha(Z) &= \mathsf{NZ} && \text{si } \emptyset \subset Z \subseteq \mathbb{Z}^\star \\
\alpha(Z) &= \mathsf{POS} && \text{si } \emptyset \subset Z \subseteq \mathbb{Z}^+ = \mathbb{N} \\
\alpha(Z) &= \mathsf{NEG} && \text{si } \emptyset \subset Z \subseteq \mathbb{Z}^- \\
\alpha(Z) &= \mathsf{SPOS} && \text{si } \emptyset \subset Z \subseteq \mathbb{Z}^{+\star} = \mathbb{N}^\star \\
\alpha(Z) &= \mathsf{SNEG} && \text{si } \emptyset \subset Z \subseteq \mathbb{Z}^{-\star} \\
\alpha(Z) &= \mathsf{INT} && \text{si erreur} \notin Z \supset \emptyset \\
\alpha(\{\text{erreur}\}) &= \mathsf{Erreur} \\
\alpha(\emptyset) &= \bot \\
\alpha(Z) &= \top && \text{si erreur} \in Z \text{ et } \emptyset \subset Z \setminus \{\text{erreur}\} \subseteq \mathbb{Z}
\end{aligned}$$

Opérateur de concrétisation

$$\begin{aligned}
\gamma &: \textit{Signes} \to \mathcal{P}(\mathbb{Z}) \cup \{\text{erreur}\} \\
\gamma(\top) &= \mathbb{Z} \cup \{\text{erreur}\} \\
\gamma(\text{INT}) &= \mathbb{Z} \\
\gamma(\text{POS}) &= \{v \in \mathbb{Z} \mid v \geq 0\} \\
\gamma(\text{NEG}) &= \{v \in \mathbb{Z} \mid v \leq 0\} \\
\gamma(\text{SPOS}) &= \{v \in \mathbb{Z} \mid v > 0\} \\
\gamma(\text{SNEG}) &= \{v \in \mathbb{Z} \mid v < 0\} \\
\gamma(\text{Z}) &= \{0\} \\
\gamma(\text{NZ}) &= \{v \in \mathbb{Z} \mid v \neq 0\} \\
\gamma(\bot) &= \emptyset
\end{aligned}$$

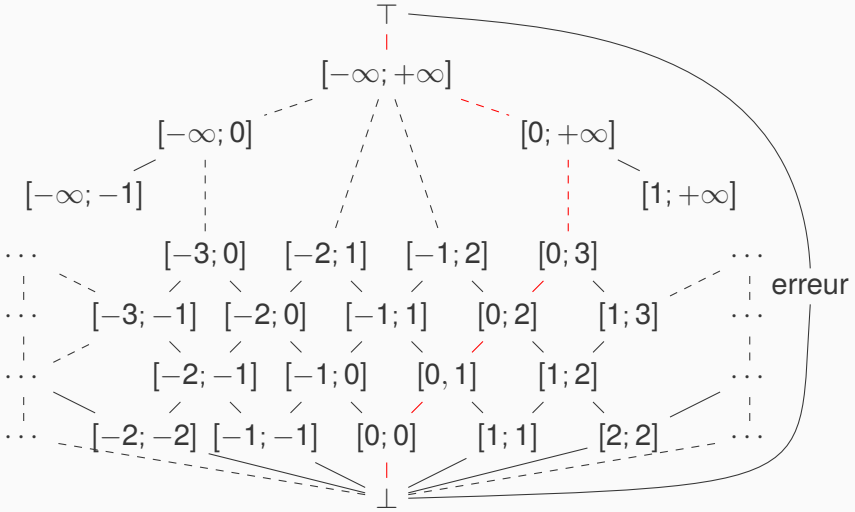
Lemme

(α, γ) est une insertion de Galois.

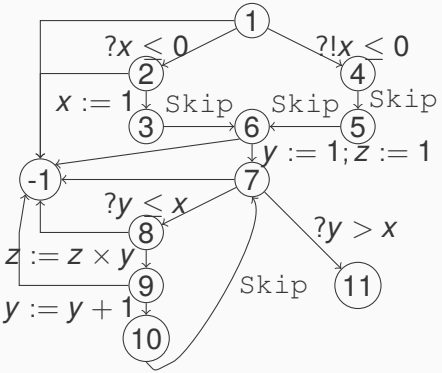
Interprétation

- on peut définir une interprétation abstraite de la sémantique collectrice
- correcte par construction :-)
- termine dans le cas du treillis des signes (treillis fini)
- ne termine pas en présence de boucle si le treillis ne vérifie pas la condition de chaîne croissante :-(

Domaine des intervalles

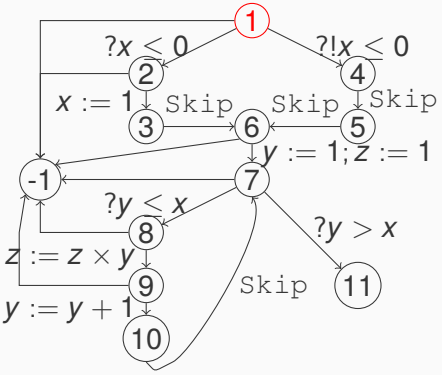


Position du problème



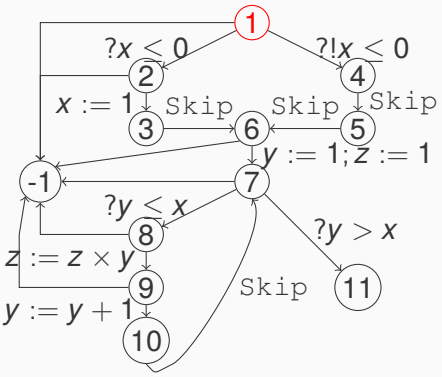
$$R(s_1)=\bot^\sharp$$
$$R(s_2)=\bot^\sharp$$
$$R(s_3)=\bot^\sharp$$
$$R(s_4)=\bot^\sharp$$
$$R(s_5)=\bot^\sharp$$
$$R(s_6)=\bot^\sharp$$
$$R(s_7)=\bot^\sharp$$
$$R(s_8)=\bot^\sharp$$
$$R(s_9)=\bot^\sharp$$
$$R(s_{10})=\bot^\sharp$$
$$R(s_{11})=\bot^\sharp$$
$$R(s_{-1})=\bot^\sharp$$

Position du problème



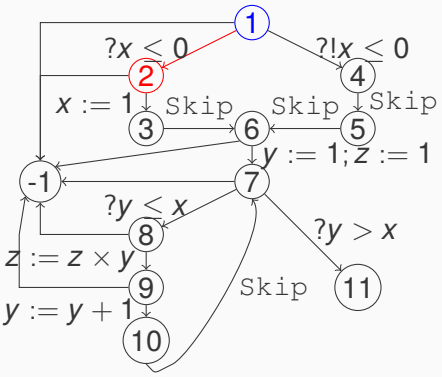
$R(s_1) = \bot^\sharp$
$R(s_2) = \bot^\sharp$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



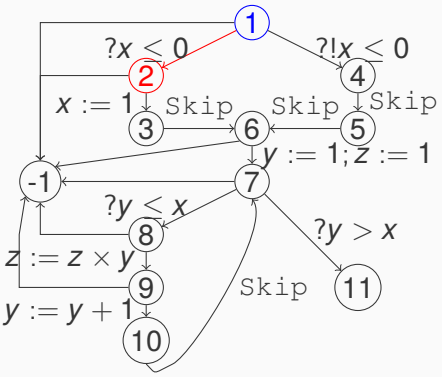
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = \bot^\sharp$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



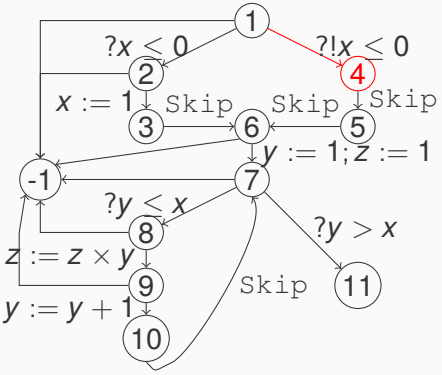
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = \bot^\sharp$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



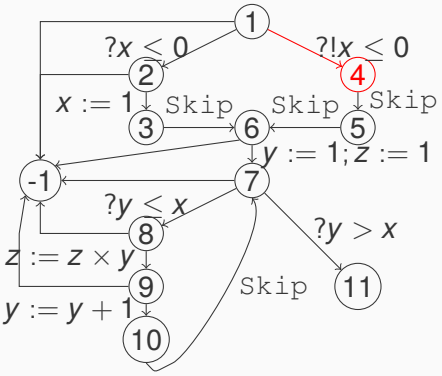
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



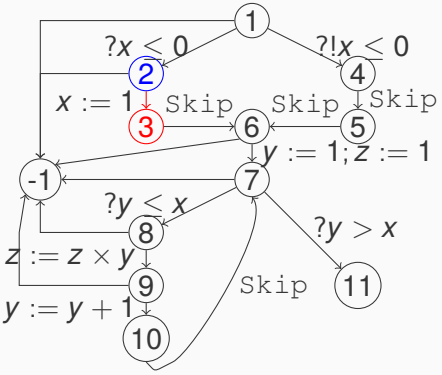
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



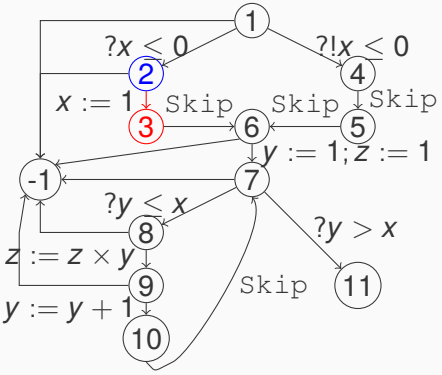
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



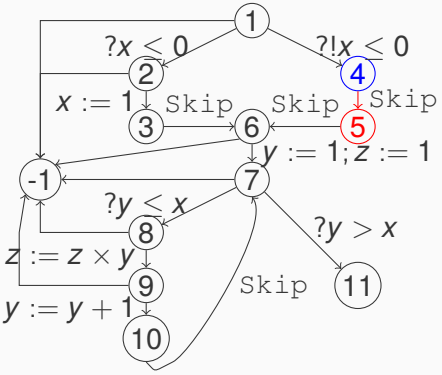
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



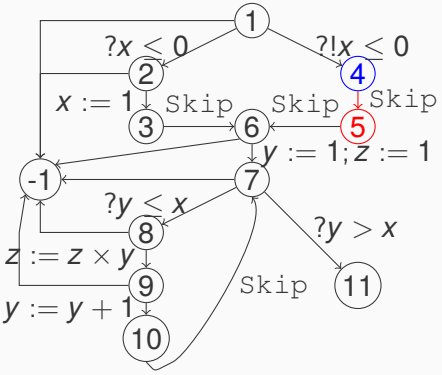
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



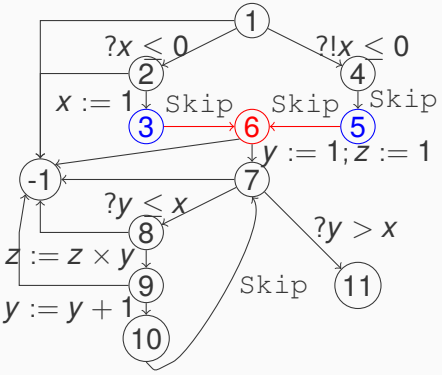
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



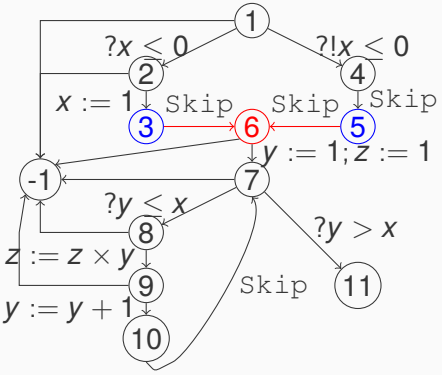
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$\bot^\sharp$
$R(s_7)$=$\bot^\sharp$
$R(s_8)$=$\bot^\sharp$
$R(s_9)$=$\bot^\sharp$
$R(s_{10})$=$\bot^\sharp$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Position du problème



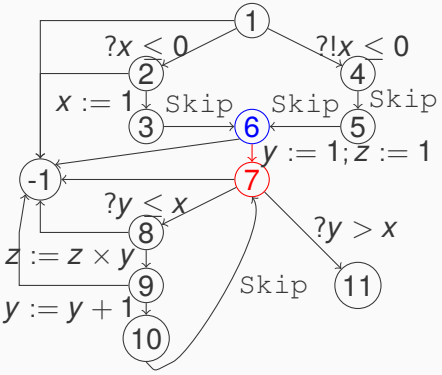
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



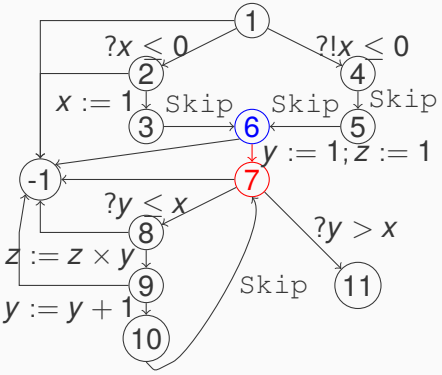
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



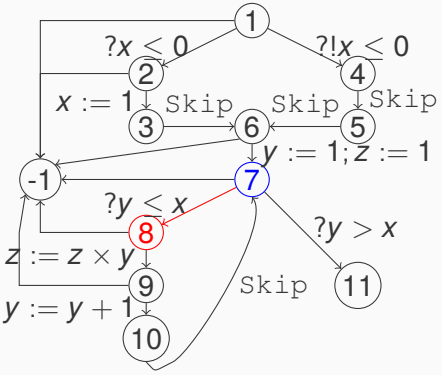
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



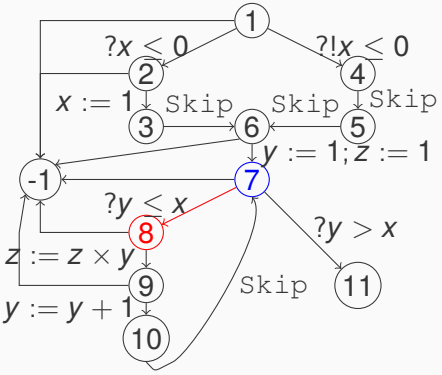
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



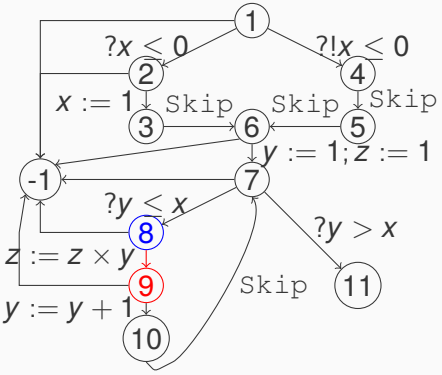
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



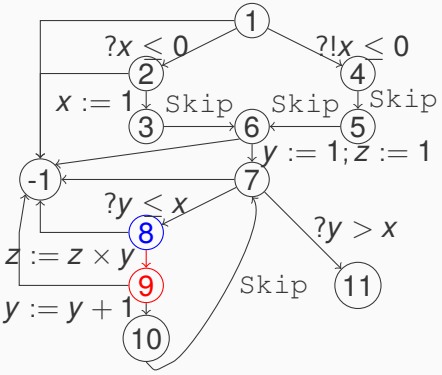
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



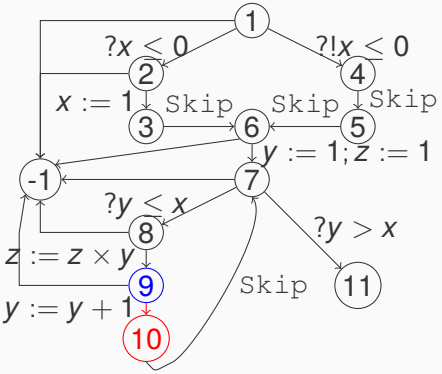
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



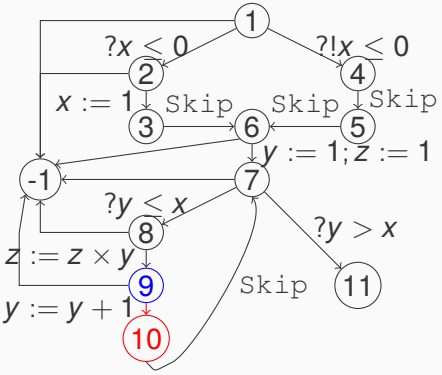
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$\bot^\sharp$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Position du problème



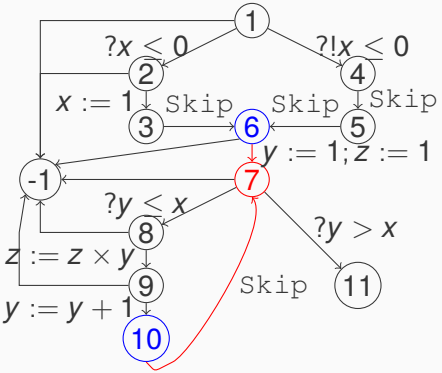
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



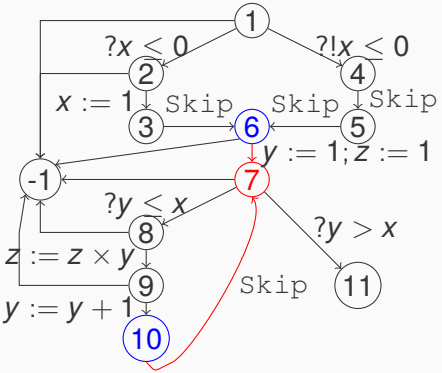
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



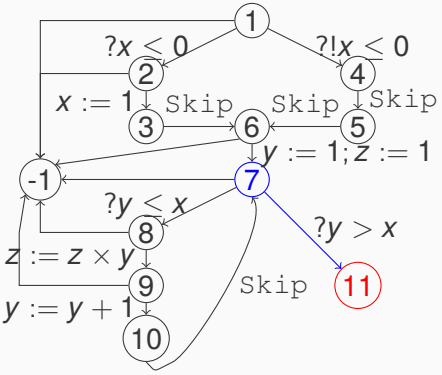
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Position du problème



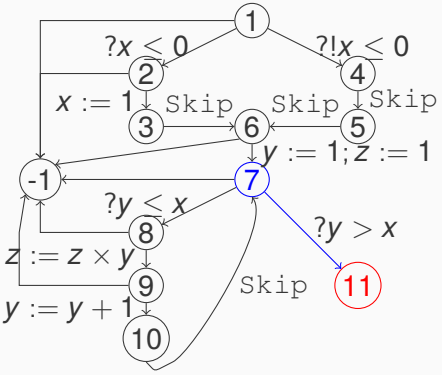
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



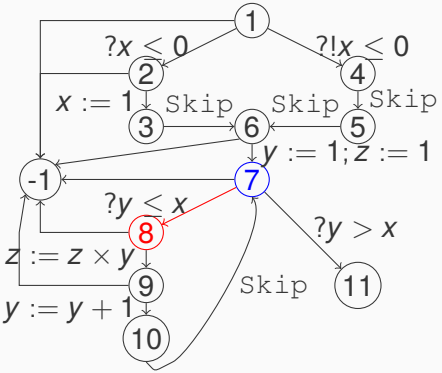
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Position du problème



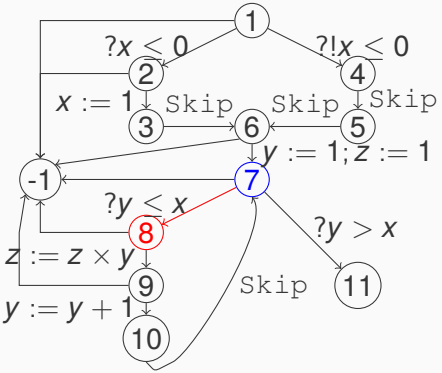
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



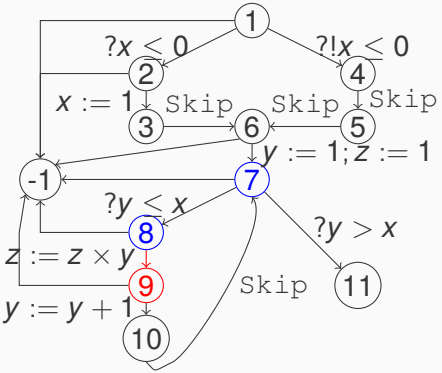
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



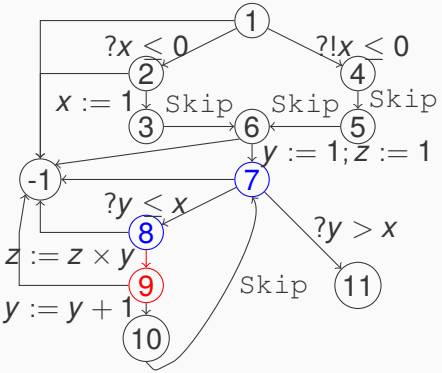
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



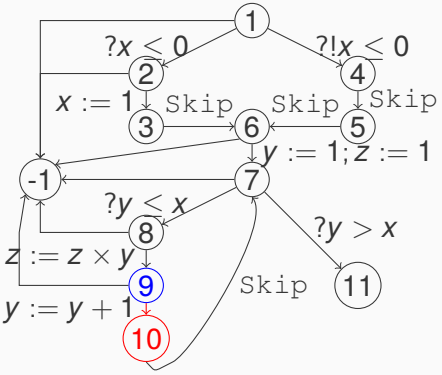
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



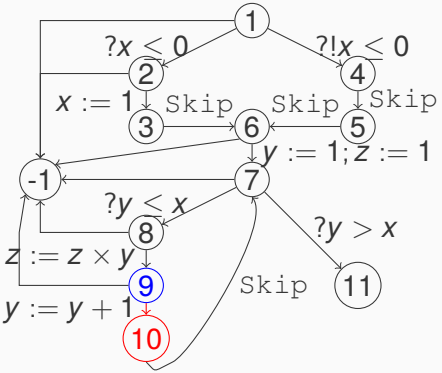
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Position du problème



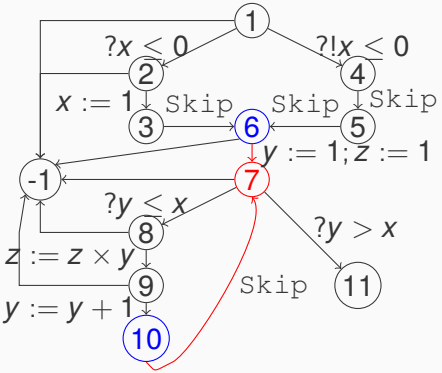
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



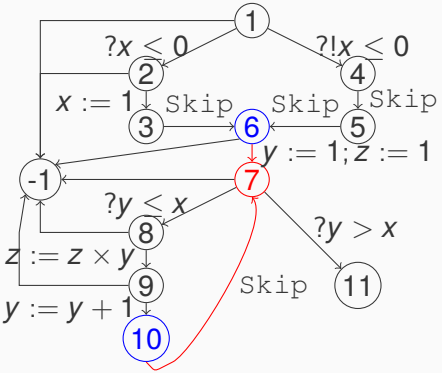
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10})$=$y \leftarrow [2; 3]; z \leftarrow [1; 2]$
$R(s_{11})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Position du problème



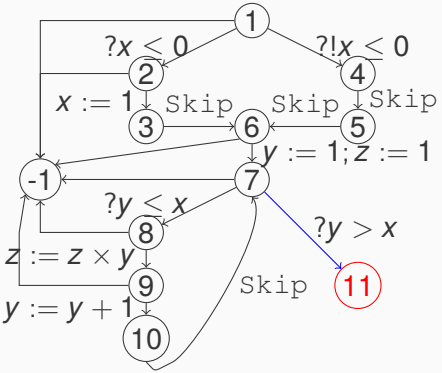
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10}) = y \leftarrow [2; 3]; z \leftarrow [1; 2]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



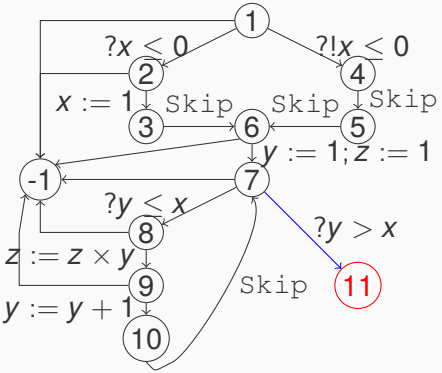
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 3]; z \leftarrow [1; 2]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10}) = y \leftarrow [2; 3]; z \leftarrow [1; 2]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



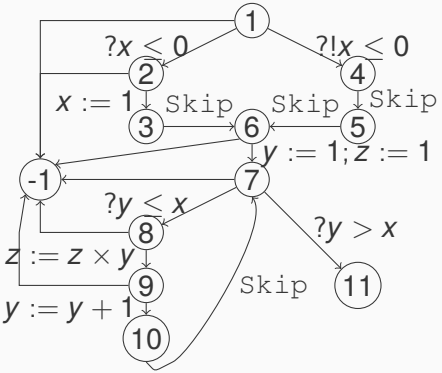
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 3]; z \leftarrow [1; 2]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10}) = y \leftarrow [2; 3]; z \leftarrow [1; 2]$
$R(s_{11}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 3]; z \leftarrow [1; 2]$
$R(s_8) = y \leftarrow [1; 2]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 2]; z \leftarrow [1; 2]$
$R(s_{10}) = y \leftarrow [2; 3]; z \leftarrow [1; 2]$
$R(s_{11}) = y \leftarrow [2, 3]; z \leftarrow [1; 2]$
$R(s_{-1}) = \bot^\sharp$

Position du problème



$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8) = y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_9) = y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10}) = y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11}) = y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{-1}) = \bot^\sharp$

Vue informelle

On veut obtenir une analyse qui termine toujours, même quand le treillis ne possède pas la propriété de chaîne ascendante.

Pour cela, on va utiliser une sur-approximation de la fonction de transition abstraite $F^\sharp$, construite de manière à garantir la terminaison.

Retour sur l'analyse flot de données

Notre analyse consiste à calculer les itérations successives de $F^\sharp$ en partant de $\bot^\sharp$. Avec $R_n^\sharp = {F^\sharp}^n(\bot^\sharp)$, on a $R_n^\sharp \sqsubseteq^\sharp R_{n+1}^\sharp$, et il vient :

$$R_{n+1}^\sharp \quad = \quad F^\sharp(R_n^\sharp)$$

Retour sur l'analyse flot de données

Notre analyse consiste à calculer les itérations successives de $F^\sharp$ en partant de $\bot^\sharp$.
Avec $R^\sharp_n = {F^\sharp}^n(\bot^\sharp)$, on a $R^\sharp_n \sqsubseteq^\sharp R^\sharp_{n+1}$, et il vient :

$$\begin{aligned} R^\sharp_{n+1} &= F^\sharp(R^\sharp_n) \\ &= \begin{cases} R^\sharp_n & \text{si } F(R^\sharp_n) \sqsubseteq^\sharp R^\sharp_n \\ R^\sharp_n \sqcup^\sharp F^\sharp(R^\sharp_n) & \text{sinon} \end{cases} \end{aligned}$$

Notre analyse consiste à calculer les itérations successives de $F^\sharp$ en partant de $\bot^\sharp$. Avec $R_n^\sharp = {F^\sharp}^n(\bot^\sharp)$, on a $R_n^\sharp \sqsubseteq^\sharp R_{n+1}^\sharp$, et il vient :

$$\begin{aligned} R_{n+1}^\sharp &= F^\sharp(R_n^\sharp) \\ &= \begin{cases} R_n^\sharp & \text{si } F(R_n^\sharp) \sqsubseteq^\sharp R_n^\sharp \\ R_n^\sharp \sqcup^\sharp F^\sharp(R_n^\sharp) & \text{sinon} \end{cases} \end{aligned}$$

Retrouver la terminaison

Dans cette présentation, on dispose d'une nouvelle possibilité pour faire une approximation du point fixe : remplacer $\sqcup^\sharp$ par un opérateur donnant un autre majorant. Un tel opérateur s'appelle Opérateur d'élargissement.

Définition : opérateur d'élargissement $\triangledown$ sur un treillis $(L, \sqsubseteq)$

Opérateur binaire vérifiant les propriétés suivantes :

- $\forall x, y \in L, x \sqsubseteq x \triangledown y \wedge y \sqsubseteq x \triangledown y$ (majorant de ses arguments)
- Pour toute chaîne ascendante x_n, la chaîne $y_n = y_{n-1} \triangledown x_n$ est stationnaire.

Théorème

Pour $\triangledown$ et une fonction monotone F, la chaîne R_n définie par

$$\begin{aligned} R_0 &= \bot^\sharp \\ R_{n+1} &= \begin{cases} R_n & \text{si } F(R_n) \sqsubseteq R_n \\ R_n \triangledown F(R_n) & \text{sinon} \end{cases} \end{aligned}$$

est stationnaire à partir d'un certain rang N. De plus $\text{lfp}(F) \sqsubseteq R_N$

Exemple d'opérateur

Pour le treillis des intervalles, un exemple simple d'opérateur d'élargissement est le suivant :

$$
\begin{aligned}
\bot^\sharp \triangledown v^\sharp &= v^\sharp \\
v^\sharp \triangledown \bot^\sharp &= v^\sharp \\
\top^\sharp \triangledown v^\sharp &= \top^\sharp \\
v^\sharp \triangledown \top^\sharp &= \top^\sharp
\end{aligned}
$$

$$
[i^1_{min}; i^1_{max}] \triangledown [i^2_{min}; i^2_{max}] = [i_{min}; i_{max}] \quad \text{avec} \begin{cases} i_{min} = i^1_{min} & \text{si } i^1_{min} \leq i^2_{min} \\ \quad = -\infty & \text{sinon} \\ i_{max} = i^1_{max} & \text{si } i^1_{max} \geq i^2_{max} \\ \quad = +\infty & \text{sinon} \end{cases}
$$

Remarque

Un opérateur d'élargissement n'est en général pas commutatif.

Exemple d'opérateur(2)

On étend cet opérateur aux environnements abstraits :

$$\rho_1^\sharp \triangledown' \rho_2^\sharp = \lambda x.\rho_1^\sharp(x) \triangledown \rho_2^\sharp(x)$$

Exemple d'opérateur(2)

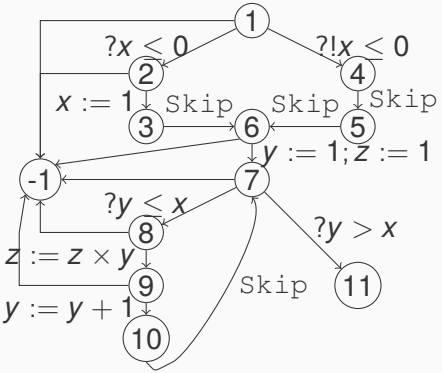
On étend cet opérateur aux environnements abstraits :

$$\rho_1^\sharp \triangledown' \rho_2^\sharp = \lambda x. \rho_1^\sharp(x) \triangledown \rho_2^\sharp(x)$$

puis à l'ensemble d'un programme. Pour cela, on peut remarquer qu'il suffit de faire l'élargissement aux points d'entrée d'une boucle pour préserver la terminaison. Avec $\mathcal{B} \subseteq \mathcal{S}$ les points correspondants, on a :

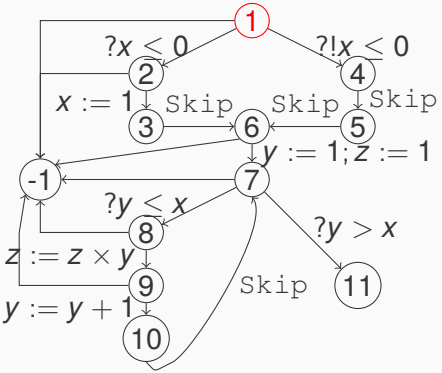
$$R_1^\sharp \triangledown'' R_2^\sharp = \lambda s. \begin{cases} R_1(s) \sqcup^\sharp R_2(s) & \text{si } s \notin \mathcal{B} \\ R_1(s) \triangledown R_2(s) & \text{si } s \in \mathcal{B} \end{cases}$$

Exemple d'élargissement



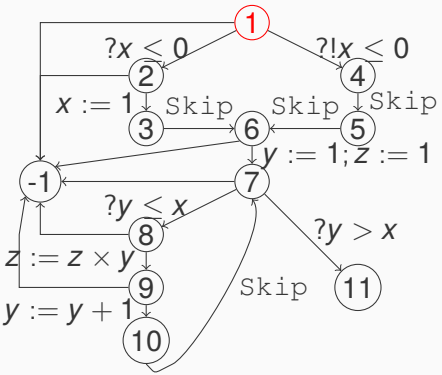
$$R(s_1) = \bot^\sharp$$
$$R(s_2) = \bot^\sharp$$
$$R(s_3) = \bot^\sharp$$
$$R(s_4) = \bot^\sharp$$
$$R(s_5) = \bot^\sharp$$
$$R(s_6) = \bot^\sharp$$
$$R(s_7) = \bot^\sharp$$
$$R(s_8) = \bot^\sharp$$
$$R(s_9) = \bot^\sharp$$
$$R(s_{10}) = \bot^\sharp$$
$$R(s_{11}) = \bot^\sharp$$
$$R(s_{-1}) = \bot^\sharp$$

Exemple d'élargissement



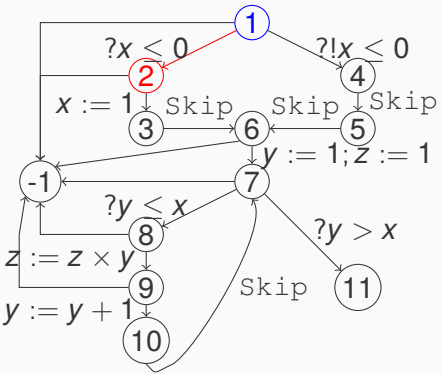
$R(s_1) = \bot^\sharp$
$R(s_2) = \bot^\sharp$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



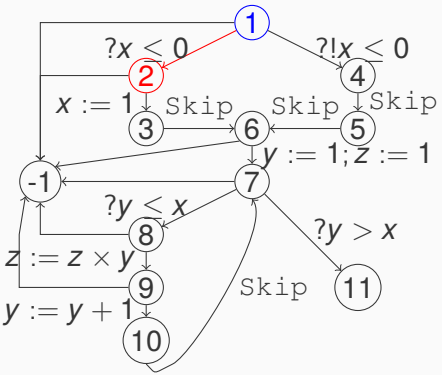
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = \bot^\sharp$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



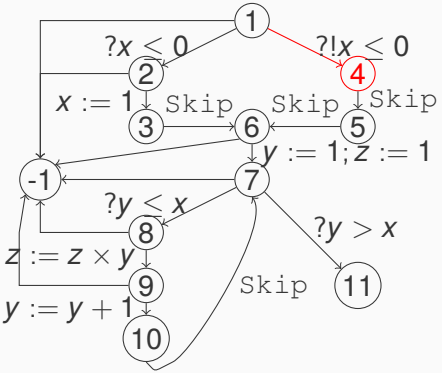
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = \bot^\sharp$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



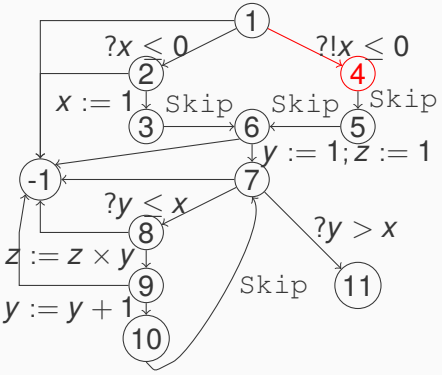
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



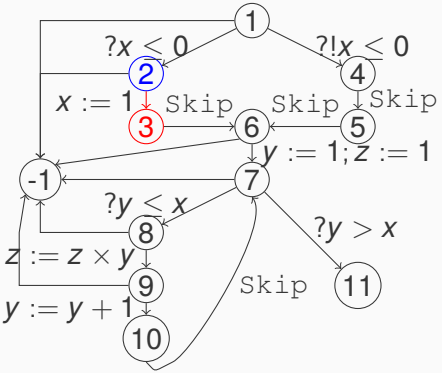
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = \bot^\sharp$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



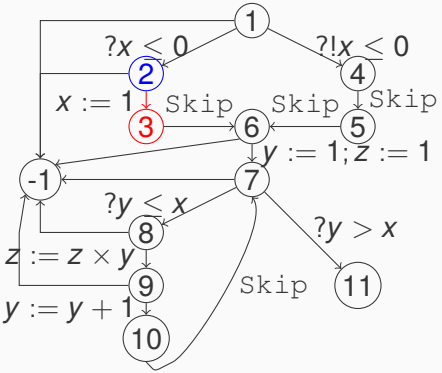
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



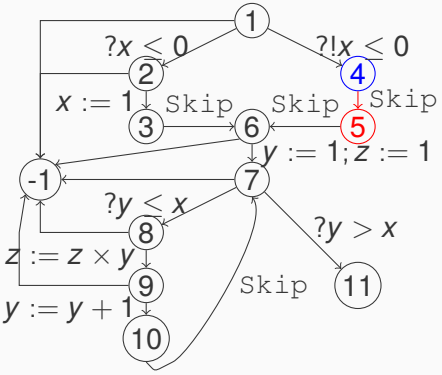
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = \bot^\sharp$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



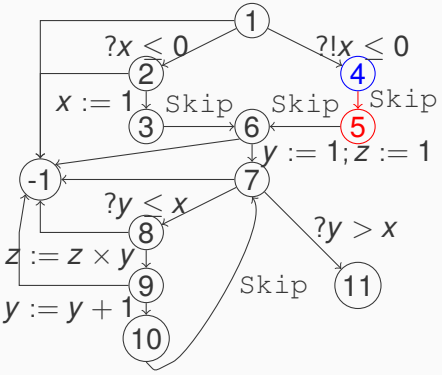
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



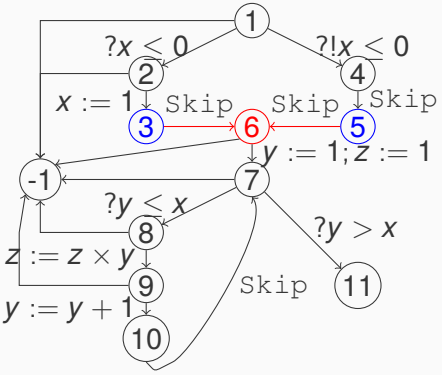
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = \bot^\sharp$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



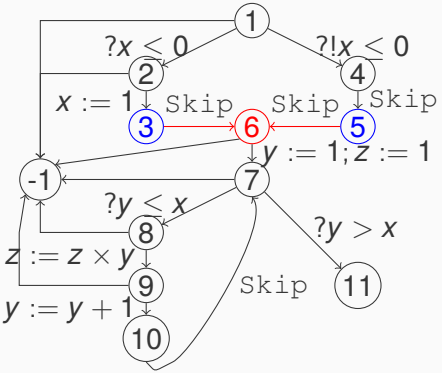
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = \bot^\sharp$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



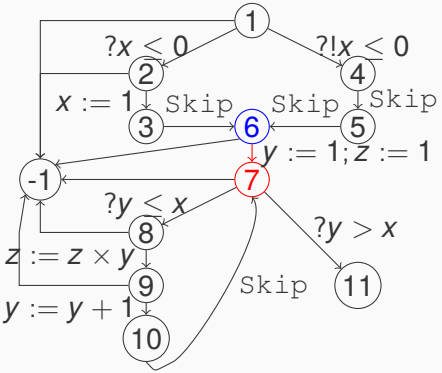
$R(s_1)=x \leftarrow [-\infty;+\infty]$
$R(s_2)=x \leftarrow [-\infty;0]$
$R(s_3)=x \leftarrow [1;1]$
$R(s_4)=x \leftarrow [1;+\infty]$
$R(s_5)=x \leftarrow [1;+\infty]$
$R(s_6)=\bot^\sharp$
$R(s_7)=\bot^\sharp$
$R(s_8)=\bot^\sharp$
$R(s_9)=\bot^\sharp$
$R(s_{10})=\bot^\sharp$
$R(s_{11})=\bot^\sharp$
$R(s_{-1})=\bot^\sharp$

Exemple d'élargissement



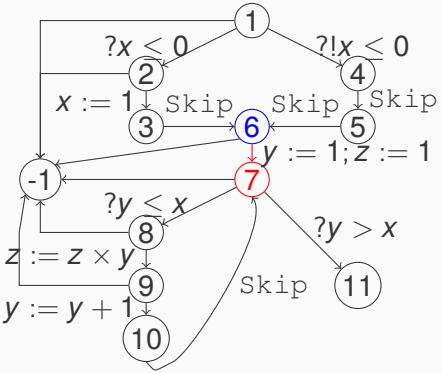
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = \bot^\sharp$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



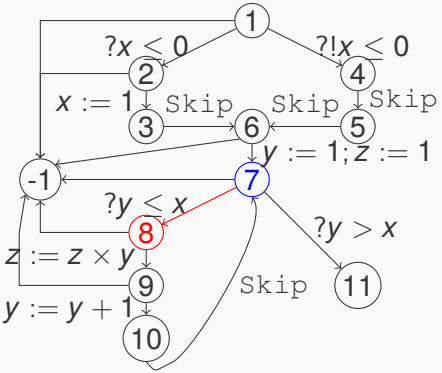
$R(s_1)=x \leftarrow [-\infty;+\infty]$
$R(s_2)=x \leftarrow [-\infty;0]$
$R(s_3)=x \leftarrow [1;1]$
$R(s_4)=x \leftarrow [1;+\infty]$
$R(s_5)=x \leftarrow [1;+\infty]$
$R(s_6)=x \leftarrow [1;+\infty]$
$R(s_7)=\bot^\sharp$
$R(s_8)=\bot^\sharp$
$R(s_9)=\bot^\sharp$
$R(s_{10})=\bot^\sharp$
$R(s_{11})=\bot^\sharp$
$R(s_{-1})=\bot^\sharp$

Exemple d'élargissement



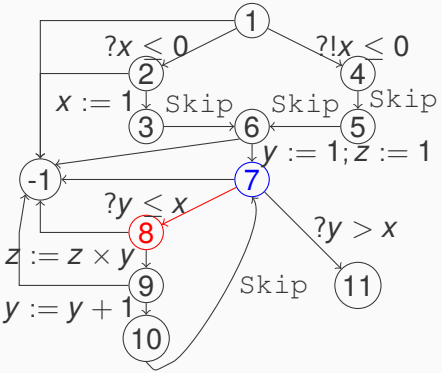
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = \bot^\sharp$
$R(s_9) = \bot^\sharp$
$R(s_{10}) = \bot^\sharp$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



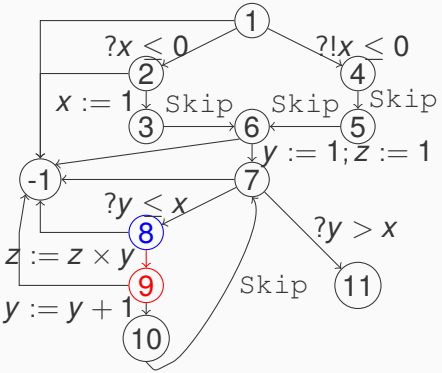
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8)$=$\bot^\sharp$
$R(s_9)$=$\bot^\sharp$
$R(s_{10})$=$\bot^\sharp$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



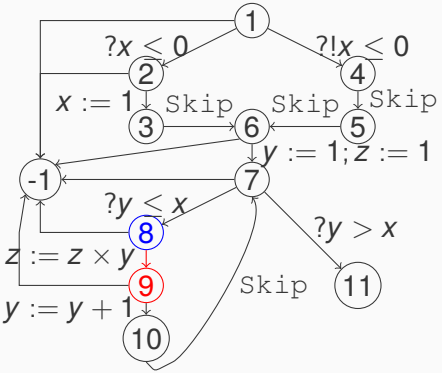
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$\bot^\sharp$
$R(s_{10})$=$\bot^\sharp$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



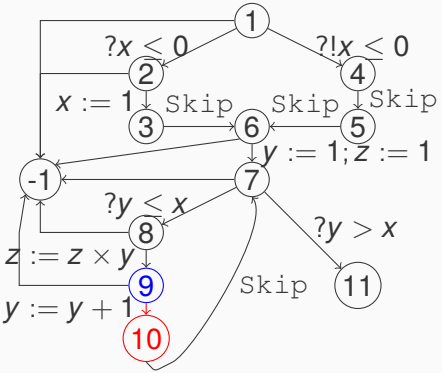
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$\bot^\sharp$
$R(s_{10})$=$\bot^\sharp$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



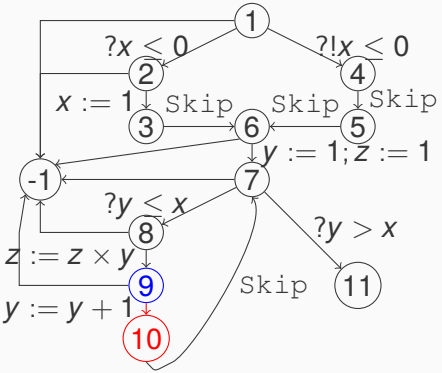
$R(s_1)=x \leftarrow [-\infty;+\infty]$
$R(s_2)=x \leftarrow [-\infty;0]$
$R(s_3)=x \leftarrow [1;1]$
$R(s_4)=x \leftarrow [1;+\infty]$
$R(s_5)=x \leftarrow [1;+\infty]$
$R(s_6)=x \leftarrow [1;+\infty]$
$R(s_7)=y \leftarrow [1;1]; z \leftarrow [1;1]$
$R(s_8)=y \leftarrow [1;1]; z \leftarrow [1;1]$
$R(s_9)=y \leftarrow [1;1]; z \leftarrow [1;1]$
$R(s_{10})=\bot^\sharp$
$R(s_{11})=\bot^\sharp$
$R(s_{-1})=\bot^\sharp$

Exemple d'élargissement



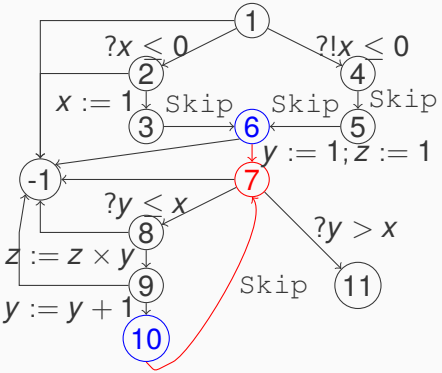
$R(s_1)=x \leftarrow [-\infty;+\infty]$
$R(s_2)=x \leftarrow [-\infty;0]$
$R(s_3)=x \leftarrow [1;1]$
$R(s_4)=x \leftarrow [1;+\infty]$
$R(s_5)=x \leftarrow [1;+\infty]$
$R(s_6)=x \leftarrow [1;+\infty]$
$R(s_7)=y \leftarrow [1;1]; z \leftarrow [1;1]$
$R(s_8)=y \leftarrow [1;1]; z \leftarrow [1;1]$
$R(s_9)=y \leftarrow [1;1]; z \leftarrow [1;1]$
$R(s_{10})=\bot^\sharp$
$R(s_{11})=\bot^\sharp$
$R(s_{-1})=\bot^\sharp$

Exemple d'élargissement



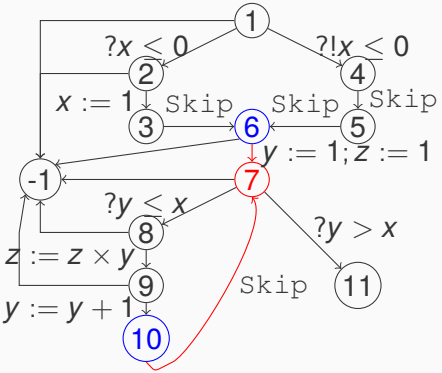
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = \bot^\sharp$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



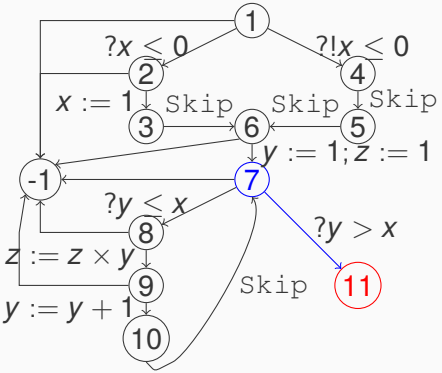
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



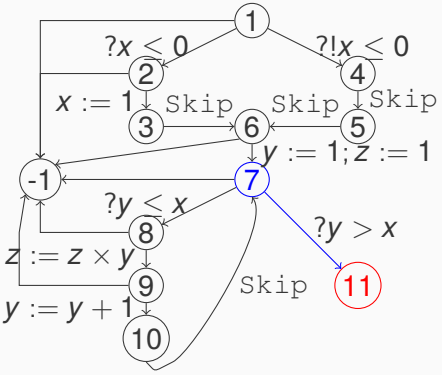
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



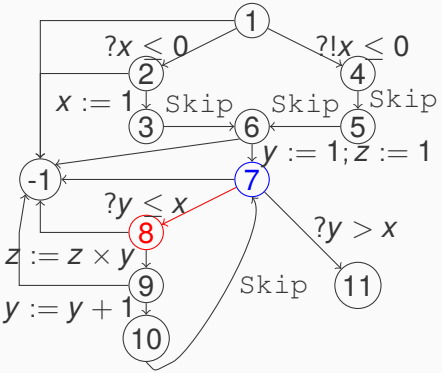
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$\bot^\sharp$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



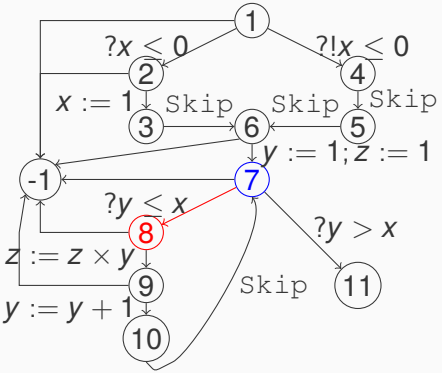
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



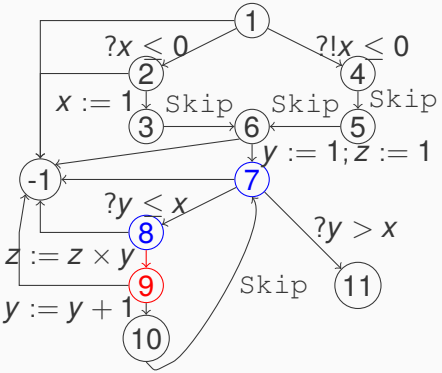
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



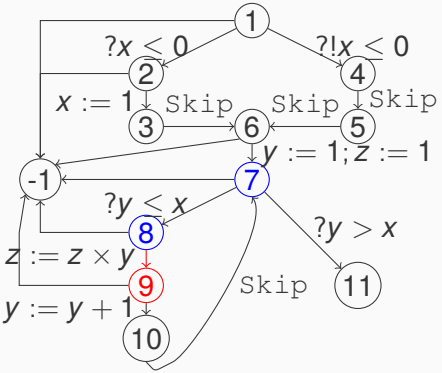
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; 1]; z \leftarrow [1; 1]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



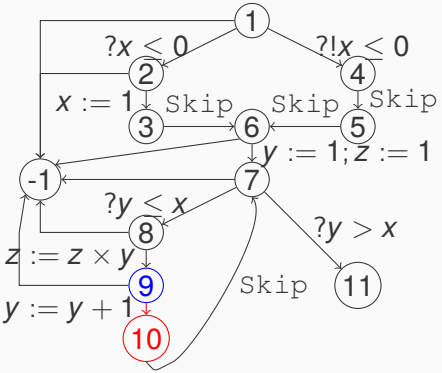
$$
\begin{aligned}
R(s_1) &= x \leftarrow [-\infty; +\infty] \\
R(s_2) &= x \leftarrow [-\infty; 0] \\
R(s_3) &= x \leftarrow [1; 1] \\
R(s_4) &= x \leftarrow [1; +\infty] \\
R(s_5) &= x \leftarrow [1; +\infty] \\
R(s_6) &= x \leftarrow [1; +\infty] \\
R(s_7) &= y \leftarrow [1; +\infty]; z \leftarrow [1; 1] \\
R(s_8) &= y \leftarrow [1; +\infty]; z \leftarrow [1; 1] \\
R(s_9) &= y \leftarrow [1; 1]; z \leftarrow [1; 1] \\
R(s_{10}) &= y \leftarrow [2; 2]; z \leftarrow [1; 1] \\
R(s_{11}) &= y \leftarrow [2; +\infty]; z \leftarrow [1; 1] \\
R(s_{-1}) &= \bot^\sharp
\end{aligned}
$$

Exemple d'élargissement



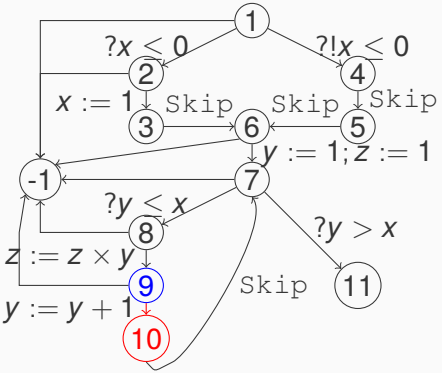
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



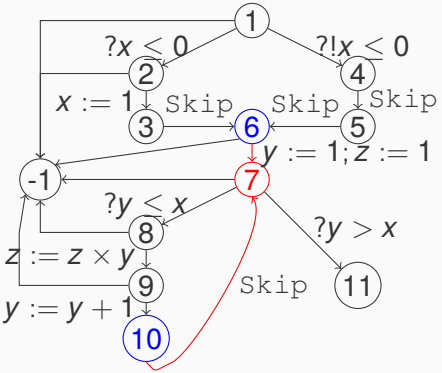
$R(s_1) = x \leftarrow [-\infty; +\infty]$
$R(s_2) = x \leftarrow [-\infty; 0]$
$R(s_3) = x \leftarrow [1; 1]$
$R(s_4) = x \leftarrow [1; +\infty]$
$R(s_5) = x \leftarrow [1; +\infty]$
$R(s_6) = x \leftarrow [1; +\infty]$
$R(s_7) = y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8) = y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9) = y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10}) = y \leftarrow [2; 2]; z \leftarrow [1; 1]$
$R(s_{11}) = y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1}) = \bot^\sharp$

Exemple d'élargissement



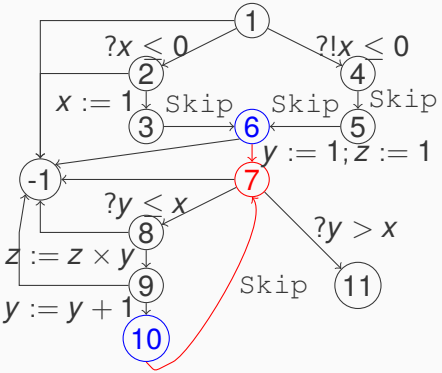
$$
\begin{aligned}
R(s_1) &= x \leftarrow [-\infty; +\infty] \\
R(s_2) &= x \leftarrow [-\infty; 0] \\
R(s_3) &= x \leftarrow [1; 1] \\
R(s_4) &= x \leftarrow [1; +\infty] \\
R(s_5) &= x \leftarrow [1; +\infty] \\
R(s_6) &= x \leftarrow [1; +\infty] \\
R(s_7) &= y \leftarrow [1; +\infty]; z \leftarrow [1; 1] \\
R(s_8) &= y \leftarrow [1; +\infty]; z \leftarrow [1; 1] \\
R(s_9) &= y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty] \\
R(s_{10}) &= y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty] \\
R(s_{11}) &= y \leftarrow [2; +\infty]; z \leftarrow [1; 1] \\
R(s_{-1}) &= \bot^\sharp
\end{aligned}
$$

Exemple d'élargissement



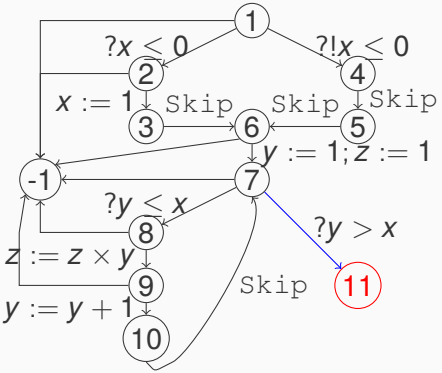
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



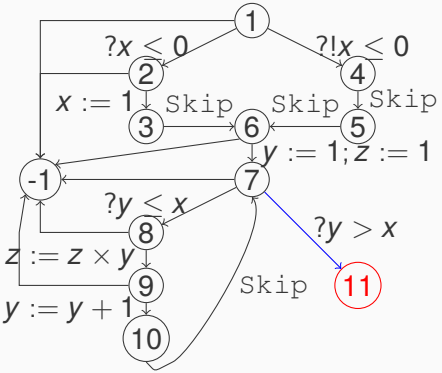
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



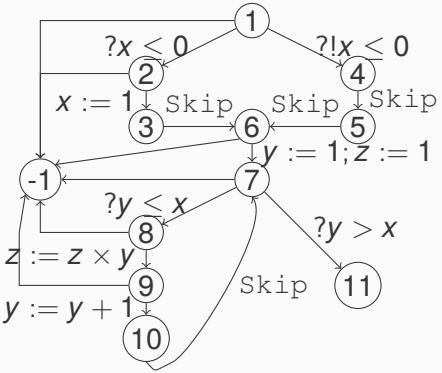
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; 1]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



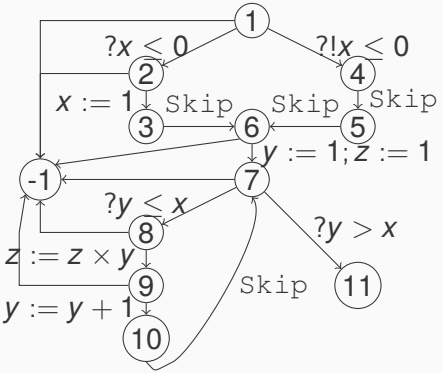
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2, +\infty]; z \leftarrow [1; +\infty]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



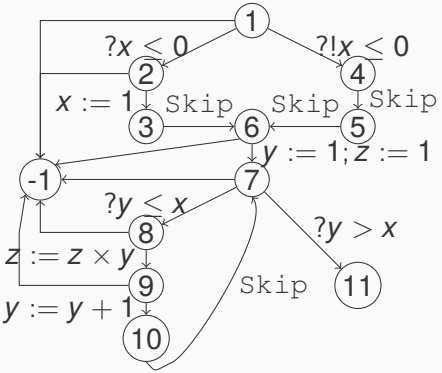
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; 1]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2, +\infty]; z \leftarrow [1; +\infty]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



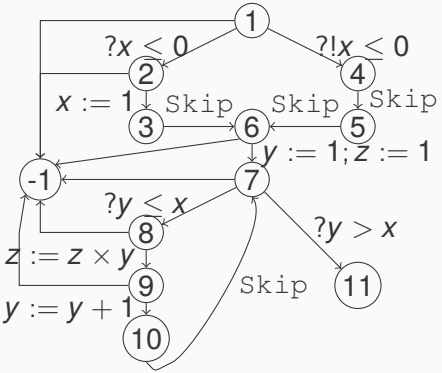
$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2, +\infty]; z \leftarrow [1; +\infty]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2, +\infty]; z \leftarrow [1; +\infty]$
$R(s_{-1})$=$\bot^\sharp$

Exemple d'élargissement



$R(s_1)$=$x \leftarrow [-\infty; +\infty]$
$R(s_2)$=$x \leftarrow [-\infty; 0]$
$R(s_3)$=$x \leftarrow [1; 1]$
$R(s_4)$=$x \leftarrow [1; +\infty]$
$R(s_5)$=$x \leftarrow [1; +\infty]$
$R(s_6)$=$x \leftarrow [1; +\infty]$
$R(s_7)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_8)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_9)$=$y \leftarrow [1; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{10})$=$y \leftarrow [2; +\infty]; z \leftarrow [1; +\infty]$
$R(s_{11})$=$y \leftarrow [2, +\infty]; z \leftarrow [1; +\infty]$
$R(s_{-1})$=$\bot^\sharp$

Ouvrages

- Hanne Nielson, Flemming Nielson, et Chris Hankin. *Principles of Program Analysis*. Springer Verlag 1999
- Neil Jones et Flemming Nielson. *Abstract Interpretation : a Semantics-Based Tool for Program Analysis*. Handbook of Logic in Computer Science (vol. 4) 1994

Fondations

- Patrick et Radhia Cousot. *Abstract Interpretation : a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. PoPL'77

Partie VI

Méthode déductive

Comment prouver la correction ?

Quelles garanties ?

Procédure

- raisonne par déduction sur le programme de manière automatique
- prend en compte les spécifications formelles

Obligation de preuve

- extraire les propriétés à vérifier (appelées obligations de preuve - OP)
- la vérification des OP est laissée à la charge de l'utilisateur
- l'utilisation de prouveurs automatiques ou semi-automatiques est indispensable

Garantie

Toutes les OP vérifiées = programme correct vis-à-vis de sa spécification

Triplet de Hoare

- Un triplet de Hoare est un triplet noté $\{P\}i\{Q\}$ où P et Q sont des propriétés (formules logiques) et i est une instruction du programme.
- P est appelé la précondition de i
- Q est appelé la postcondition de i
- Informellement, un triplet de Hoare est valide si la propriété suivante est vraie :

  si la précondition P est valide avant d'exécuter l'instruction i, alors la postcondition Q sera valide après l'exécution de i

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\}x := x + 1\{x = 11\}$?

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\} x := x + 1 \{x = 11\}$? ✓
- $\{c = 0\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 0 \wedge c = 0\}$?

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\} x := x + 1 \{x = 11\}$? ✓
- $\{c = 0\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 0 \wedge c = 0\}$? ×
- $\{c = 2\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 2 \wedge c = 2\}$?

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\} x := x + 1 \{x = 11\}$? ✓
- $\{c = 0\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 0 \wedge c = 0\}$? ×
- $\{c = 2\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 2 \wedge c = 2\}$? ✓
- $\{\text{false}\} x := 0; y := 1 \{x = 0 \wedge y = 2\}$?

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\} x := x + 1 \{x = 11\}$? ✓
- $\{c = 0\}$ `if` c `then` $x := c$ `else` $x = 1$ `fi` $\{x = 0 \wedge c = 0\}$? ×
- $\{c = 2\}$ `if` c `then` $x := c$ `else` $x = 1$ `fi` $\{x = 2 \wedge c = 2\}$? ✓
- $\{\text{false}\} x := 0; y := 1 \{x = 0 \wedge y = 2\}$? ✓
- $\{i = 1\}$ `while` $i <= 10$ `do` $i := i + 1$ `end` $\{i = 10\}$?

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\} x := x + 1 \{x = 11\}$? ✓
- $\{c = 0\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 0 \wedge c = 0\}$? ×
- $\{c = 2\} \texttt{if } c \texttt{ then } x := c \texttt{ else } x = 1 \texttt{ fi} \{x = 2 \wedge c = 2\}$? ✓
- $\{\text{false}\} x := 0; y := 1 \{x = 0 \wedge y = 2\}$? ✓
- $\{i = 1\} \texttt{while } i <= 10 \texttt{ do } i := i + 1 \texttt{ end} \{i = 10\}$? ×
- $\{i = 1\} \texttt{while } !(i <= 0) \texttt{ do } i := i + 1 \texttt{ end} \{i = 0\}$?

Exemples de triplets de Hoare

Les triplets de Hoare suivants sont-ils valides ?

- $\{x = 10\} x := x + 1 \{x = 11\}$? ✓
- $\{c = 0\}$ `if` c `then` $x := c$ `else` $x = 1$ `fi` $\{x = 0 \wedge c = 0\}$? ×
- $\{c = 2\}$ `if` c `then` $x := c$ `else` $x = 1$ `fi` $\{x = 2 \wedge c = 2\}$? ✓
- $\{\text{false}\} x := 0; y := 1 \{x = 0 \wedge y = 2\}$? ✓
- $\{i = 1\}$ `while` $i <= 10$ `do` $i := i + 1$ `end` $\{i = 10\}$? ×
- $\{i = 1\}$ `while` $!(i <= 0)$ `do` $i := i + 1$ `end` $\{i = 0\}$? ✓

Sémantique axiomatique

- La sémantique axiomatique d'une instruction *i* peut être définie par l'ensemble des triplets de Hoare valides portant sur *i*.
- Avec une définition informelle, il peut être néanmoins difficile de savoir si un triplet de Hoare est valide ou non, même sur un langage simple
- la sémantique axiomatique doit être définie formellement pour éviter toute ambigüité.
- On utilise un système de règles de déduction dans ce but
- Ce système est la logique de (Floyd-)Hoare

Sémantique axiomatique d'un langage simple

$$\frac{}{\{P\}\quad \texttt{Skip}\quad \{P\}} \qquad \frac{\{P\}\quad i_1\quad \{R\} \qquad \{R\}\quad i_2\quad \{Q\}}{\{P\}\quad i_1 ; i_2\quad \{Q\}} \qquad \frac{}{\{P[x \leftarrow e]\}\quad x := e\quad \{P\}}$$

$$\frac{\{P \wedge e \,!\!=\! 0\}\quad i_1\quad \{Q\} \qquad \{P \wedge e = 0\}\quad i_2\quad \{Q\}}{\{P\}\quad \texttt{if}\ e\ \texttt{then}\ i_1\ \texttt{else}\ i_2\ \texttt{fi}\quad \{Q\}}$$

$$\frac{\{I \wedge e \,!\!=\! 0\}\quad i\quad \{I\}}{\{I\}\quad \texttt{while}\ e\ \texttt{do}\ i\ \texttt{end}\quad \{I \wedge e = 0\}} \qquad \frac{P \implies P' \qquad \{P'\}\quad i\quad \{Q'\} \qquad Q' \implies Q}{\{P\}\quad i\quad \{Q\}}$$

Discussion sur la logique de Hoare

- Ces règles d'inférence définissent une sémantique, pas un algorithme
- Séquence : comment établir le prédicat R intermédiaire ?
- Boucle : comment établir l'invariant de boucle I ?
- Règle d'affaiblissement (la dernière) : quand l'appliquer ?
- Erreurs à l'exécution : comment sont-elles prises en compte ?
- Affectation : nature "arrière" de cet axiome, le secret de la réussite du calcul de plus-faible précondition qui va suivre

Exemple

Mais au tableau parce que c'est un peu pénible dans une slide

Calcul de plus-faible précondition (WP)

But

Étant donné, une précondition *Pre*, une postcondition *Post* et un programme *p*, $\{Pre\}p\{Post\}$ est-il un triplet de Hoare valide ?

Comment ?

- Le programme est vu comme un transformeur de prédicat.
- On part de la postcondition *Post* à la fin du programme et on raisonne en effectuant les étapes de calcul “à l’envers”.
- À chaque étape de calcul, on connaît la postcondition Q à vérifier et l’instruction i concernée et on déduit la propriété P minimale (la plus faible) telle que $\{P\}i\{Q\}$ soit un triplet de Hoare valide
- Lorsque le début du programme est atteint, on doit prouver $Pre \implies P$

Calcul de WP - Définition

Définition par induction sur la structure du programme

$$WP(\texttt{Skip}, Q) = Q$$

$$WP(i_1; i_2, Q) = WP(i_1, WP(i_2, Q))$$

$$WP(x := e, Q) = Q[x \leftarrow e] \qquad \text{si } e \text{ est évaluée sans erreur}$$

$$\begin{aligned} WP(\texttt{if } e \texttt{ then } i_1 \texttt{ else } i_2 \texttt{ fi}, Q) = {} & e\,!{=}0 \implies WP(i_1, Q) \\ & \wedge\, e = 0 \implies WP(i_2, Q) \end{aligned}$$

$$\begin{aligned} WP(\texttt{while } e \texttt{ do } i \texttt{ end}, Q) = {} & I \\ & \wedge\, (e\,!{=}0 \wedge I \implies WP(i, I)) \\ & \wedge\, (e = 0 \wedge I \implies Q) \end{aligned}$$

Calcul de WP - Discussion

- Obligation de preuves pour la gestion des erreurs
- Boucle : comment établir l'invariant I ?
  - Indiqué manuellement par l'utilisateur
  - Idéalement, déduit automatiquement (par exemple, par interprétation abstraite)
- *WP* définit un algorithme, modulo le problème des invariants de boucle

Corrections partielle et totale

Correction partielle

Garantie que si le programme p termine, alors p satisfait la spécification S

Correction totale

Garantie que le programme p termine et p satisfait la spécification S

Terminaison

Prouver la terminaison nécessite la donnée d'une mesure décroissante pour chaque boucle (un variant), le plus souvent indiquée manuellement par l'utilisateur

Choix

Dépend du contexte : la correction partielle peut être suffisante.

Obligations de preuve - Génération

Quand et pourquoi sont-elles générées ?

- pour garantir la sûreté du programme, dès qu'une opération peut entraîner une erreur à l'exécution
  - divisions par zéro
  - déréférencement des pointeurs
  - *overflow* des opérations arithmétiques
- pour vérifier les spécifications utilisateurs
  - postconditions
  - assertions
  - invariants et variants de boucles
  - ...

Comment effectuer une preuve ?

Prouver ? Oui, mais comment ?

- Prouver à la main
  - impossible en pratique
- Prouver à l'aide d'un assistant de preuves : outil pour assister un humain dans l'activité de preuves en le guidant, en résolvant les parties triviales ou calculatoires, et en vérifiant leur validité
  - Coq, Isabelle, PVS, ...
- Prouver avec des prouveurs automatiques : outil "presse-bouton" effectuant les preuves de manière automatique
  - Alt-Ergo, CVC4, Z3, ...

Gestion des obligations de preuve

Contexte de la preuve de programmes (de taille réelle)

- les obligations de preuve sont générées automatiquement par des outils
- plusieurs (dizaine, centaines, ... de) milliers d'obligations de preuve
- plusieurs dizaines d'hypothèses par obligation de preuve (voire plus)

Faisabilité des preuves

- infaisable par un humain
- reste très lourd avec un assistant de preuves
- emploi de prouveurs automatiques indispensables
- les prouveurs automatiques ne savent pas tout faire

Greffon WP

- outil de vérification déductive de Frama-C
- basé sur (une variante de) le calcul de plus faible précondition
- vérification modulaire
- génère des POs ensuite envoyées à des prouveurs automatiques
- si toutes les POs sont prouvées le programme respecte sa spécification

Approche par contrats

- But : spécification de fonctions
- Approche : indiquer des assertions à propos des fonctions
  - Précondition supposée vraie à l'entrée (garanti par l'appelant)
  - Postcondition doit être vraie en sortie (garanti par la fonction)
- quand la précondition est fausse, rien n'est garanti

But d'un contrat :

- refléter la spécification informelle
- ne devrait pas être modifié juste pour permettre à la preuve de passer

Retour à la valeur absolue

Spécifions la fonction valeur absolue (Démo)

Lien avec les erreurs à l'exécution

- WP peut vérifier (comme Eva) l'absence d'erreurs à l'exécution
- si cela n'est pas fait, rien ne garanti que le programme fera son travail
- aucune obligation d'utiliser le même outil, on pourrait
  - utiliser Eva pour garantir cela
  - utiliser WP uniquement pour les propriétés fonctionnelles.

Autre source d'imprécision (Démo)

La clause ASSIGNS

```
assigns v1, v2, ... vN;
```

- fait partie de la postcondition
- spécifie les positions mémoire modifiées par la fonction
- rien à dire sur les variables locales
  - une fonction est autorisée à les modifier
  - une postcondition ne peut de toute façon pas en parler
- le mot clé `\nothing` indique un ensemble vide

La fonction max avec des pointeurs

Et si on a des pointeurs ? (Démo)

Rappel - les comportements

Spécification par cas

- la précondition (**`requires`**) globales s'applique à tous les cas
- la postcondition (**`assigns, ensures`**) globales s'applique à tous les cas
- les comportements définissent des contrats pour des cas particuliers
- pour chaque cas (**`behavior`**)
  - le sous domaine est défini par la clause **`assumes`**
  - la précondition du comportement est définie par **`requires`**
  - la postcondition du comportement est définie par **`assigns`**,**`ensures`**
- **`complete behaviors`** indique que les comportements couvrent les entrées
- **`disjoint behaviors`** indique que les comportements ne se recouvrent pas

Retour sur les comportements

Spécifions la fonction valeur absolue avec des comportements (Démo)

Modularité du WP calculus

Le calcul de plus faible précondition est une technique modulaire

En particulier ...

- Chaque fonction est prouvée est isolation des autres
- Lors d'un appel de fonction, l'analyseur ne va pas voir son code, on :
  - prouve que les préconditions sont validées,
  - admet que les postconditions sont validées.

Appel de fonction, exemple

Prouvons la correction de la fonction “maximum des valeurs absolues” (Démo)

Les boucles ont un nombre d'itérations variable

- seule manière de traiter les boucles : la preuve par induction
- nous avons donc besoin d'une hypothèse d'induction
  - vraie juste avant la boucle
  - vraie après $k + 1$ itérations quand elle est vraie après $k \geq 0$ itérations

Ce type de propriété est appelée un invariant de boucle

- on génère pour cela deux obligations de preuves :
  - l'invariant de boucle est vrai initialement
  - l'invariant de boucle est préservé par toute itération

Quelques conseils

Comment trouver un bon invariant ?

- Identifier les variables modifiées par la boucle
  - Ne pas connaître le nombre d'itérations empêche la déduction de leur valeur
  - Définir leur intervalle de valeur après k itérations
  - Lister les variables modifiées avec `loop assigns`
- Identifier les actions réalisées ou les propriétés garanties par la boucle
  - Quelle partie du travail est déjà réalisée après k itérations ?
  - Pourquoi la prochaine itération peut procéder comme elle le fait ?

Nécessite de l'expérience en preuve lorsque les programmes sont complexes

Mise à zéro

Remise à zéro d'un tableau (Démo)

La terminaison de programme est indécidable

- un outil ne peut trouver une borne supérieure à un nombre d'itérations
- en revanche, si on fournit une borne supérieure, on peut la vérifier
- On appelle cette borne un variant de boucle

Variant de boucle

- expression (pas un prédicat)
- un variant n'est pas unique, si V est un variant, $V + 1$ aussi
- la borne n'a pas besoin d'être précise
- astuce simple, regarder la condition de boucle
  - si l'expression est de la forme `e1 < e2`, `e2 - e1` est un bon candidat

Mise à zéro terminaison

Remise à zéro d'un tableau (Démo)

Petite subtilité sur les (in)variants

Les boucles sont vues comme :

```
/*@ loop invariant I ;
    loop variant   V ; */
while(1){
  if(!cond) break ;
}
```

On vérifie l'(in)variant quand on retombe sur l'entrée du **`while`**.
Donc pas lors d'un **`break`**.

Exemple de terminaison prématurée de boucle

Recherche dans un tableau (Démo)

Définitions globales I

Prédicats et fonctions utilisateur

```
/*@ logic integer diff{L1, L2}(int* a integer i) =
      \at(a[i], L1) - \at(a[i], L2) ;

    predicate eq{L1, L2}(int* a, integer i) =
      diff{L1, L2}(a, i) == 0;
*/
```

Lemmes et axiomes

```
/*@ lemma transitive_eq{L1, L2, L3}:
      \forall int* a, integer x ;
        eq{L1, L2}(a, i) ==> eq{L2, L3}(a, i) ==>
          eq{L1, L3}(a, i);

    axiomatic Answer {
      logic integer attltuae reads \nothing ;
      axiom the_answer: attltuae == 42 ;
    }
*/
```

Modèle mémoire

Représentation de la mémoire

- En pratique, un programme (en particulier un programme C) contient des opérations sur la mémoire de nature complexe (manipulations de pointeurs, de tableaux, structures, ...)
- Le calcul de WP doit en tenir compte

Différents modèles mémoires

- Modèle mémoire bas-niveau : mémoire vue comme un tableau de bits ou d'octets
  - simple, modélise bien la réalité
  - possibilité de vérifier plein de propriétés (y compris très bas niveaux)
  - obligations de preuve très compliquées à vérifier
- Modèle mémoire haut-niveau : vue plus abstraite de la mémoire
  - obligations de preuve plus faciles à prouver
  - plus on est haut-niveau, moins les propriétés de bas-niveaux peuvent être prouvées
- Exemple : modèle de Burstall-Bornat pour gérer les structures. Chaque champ est considéré comme séparé des autres.
- Plus généralement, on peut séparer la mémoire en différentes régions pour faciliter la gestion des pointeurs et des tableaux.
- Limitations : par exemple non-gestion des casts de pointeurs

Articles fondateurs

- Robert W. Floyd. Assigning meanings to program.
  Mathematical Aspects of Computer Science 1967.
- C. A. R. (Tony) Hoare. An axiomatic basis for computer programming.
  Communication of the ACM 1969.
- Edsger W. Dijkstra. A Disciple of Programming.
  Prentice-Hall 1976.
- Richard Bornat. Proving pointer programs in Hoare logic.
  Mathematics of Program Construction 2000.

Bibliographie : outils

- Boogie. Programmes C♯. Langage de spécification : Spec♯.
  `http://research.microsoft.com/en-us/projects/specsharp`
- Spark-2014. Programmes Ada.
  `https://www.adacore.com/about-spark`
- ESC/Java2. Programmes Java. Langage de spécification : JML.
  `https://www.cs.ucf.edu/~leavens/JML/index.shtml`
- Frama-C (WP). Programmes C. Langage de spécification : ACSL.
  `http://frama-c.com`

Partie VII

Combinaison d'outils

Intégration d'une méthodologie plus large

Tirer le meilleur de tous les mondes ... si on peut

Qu'avons-nous vu jusqu'ici ?

Méthodes de spécification

- Spécification informelle
- Spécification formelle (ACSL ici)

Méthodes de vérification

- Test (précis, mais incomplet)
- Runtime monitoring (comme le test, plus robuste, plus coûteux)
- Interprétation abstraite (complet, mais imprécis)
- Vérification déductive (complet, mais coûteux)

Loin d'être en concurrence, ces techniques se complètent.

On n'a pas un budget infini pour le projet ...

Démarche

- Montez progressivement en confiance
- Trouvez le juste milieu entre coût de vérification et confiance
- Trouvez le bon moment pour introduire un outillage

A garder en tête

- Un patron va souvent vouloir tirer le coût vers le bas
- L'industrie n'est pas toujours au courant de l'outillage existant en R&D
- La R&D n'est pas toujours prête pour l'industrie

Aide au debugging de spécification

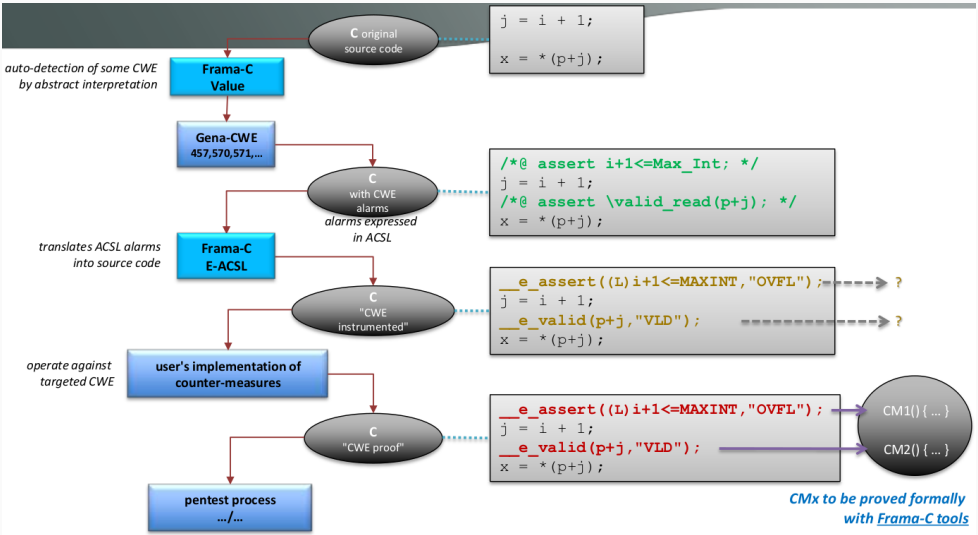
- Une spécification formelle peut être difficile à bien écrire
- On peut utiliser du runtime monitoring pour explorer la spécification

Vérifier ce qui reste à vérifier

- Des fois certaines propriétés sont trop dures à vérifier
- On peut utiliser du runtime monitoring pour construire un argumentaire

Mais on ne veut pas explorer ce qui est déjà vérifié

Voyons ça ...


C original source code
j = i + 1;
x = *(p+j);
auto-detection of some CWE by abstract interpretation
Frama-C Value
Gena-CWE 457,570,571,...
C with CWE alarms
alarms expressed in ACSL
/*@ assert i+1<=Max_Int; */
j = i + 1;
/*@ assert \valid_read(p+j); */
x = *(p+j);
translates ACSL alarms into source code
Frama-C E-ACSL
C "CWE instrumented"
__e_assert((L)i+1<=MAXINT,"OVFL");
j = i + 1;
__e_valid(p+j,"VLD");
x = *(p+j);
?
?
operate against targeted CWE
user's implementation of counter-measures
C "CWE proof"
__e_assert((L)i+1<=MAXINT,"OVFL");
j = i + 1;
__e_valid(p+j,"VLD");
x = *(p+j);
CM1(){ ... }
CM2(){ ... }
pentest process .../...
CMx to be proved formally with Frama-C tools

Pour aider la méthode déductive ...

- Vérifier l'absence d'erreur à l'exécution peut être long avec WP
- Mais Eva est très efficace pour ce travail

⇒ Eva pour vérifier l'absence d'erreur à l'exécution puis WP pour le reste

Pour aider l'interprétation abstraite

- Parfois Eva n'arrive pas à tout vérifier
- Mais WP peut souvent avec un peu plus d'informations

⇒ Quand Eva n'arrive pas à prouver, utiliser WP pour aider

Voyons ça ...

Journey to a RTE-free X.509 parser

Arnaud Ebalard, Patricia Mouy, and Ryad Benadjila
prenom.nom@ssi.gouv.fr

ANSSI

Abstract. C programming language is a security nightmare. It is error-prone and unsafe, but, year after year, the conclusion remains the same: no credible alternative will replace C in a foreseeable future; all the more in low-level developments or for constrained environments

https://github.com/ANSSI-FR/x509-parser

Partie VIII

Conclusion

A-t-on nos réponses ?

Aller plus loin ?

But initial

avoir confiance dans les logiciels embarqués critiques

Qu'avons nous mis en place ?

- du test systématique
- de l'analyse statique correcte
- de la vérification déductive
- de la vérification à l'exécution

Loin d'être en concurrence, ces techniques se complètent.

Résultat

Peut on avoir confiance dans un programme vérifié ?

- à 100%, non !
- les méthodes correctes restent un niveau d'assurance très élevé

D'où provient la confiance et la défiance ?

Nombreuses possibilités de bugs

- Implémentation des analyseurs
- Prouveurs et assistants de preuve
- Compilateur générant du code erroné
- Matériel

Solutions possibles

- noyau des assistants de preuve vérifié (Coq)
- noyau des prouveurs automatiques vérifié (Alt-Ergo)
- possibilité de vérifier les preuves des prouveurs (Zénon)
- compilateur certifié (CompCert)
- analyseur certifié (Verasco)

Mais encore ?

Autres risques

- Non-vérification des hypothèses implicites des analyseurs
- Erreur dans les spécifications
- Erreur dans une axiomatique utilisée (théorie incohérente)

Solutions possibles : réduire les risques

- vérifier toutes les hypothèses implicites
- être prudent, mettre ceintures et bretelles

Résumons

Propriétés du code

- Propriétés fonctionnelles
- Absence d'erreur d'exécution
- Terminaison
- Dépendances
- Non-interférence
- Propriétés temporelles
- ...

Périmètre de la vérification

- Quelle partie du code sous analyse ?
- Que contexte initial ?

Base de confiance

- Axiomes ACSL
- Hypothèses des analyseurs
- Doublures de fonctions
- Frama-C et prouveurs externes
- Et aussi : le compilateur, le runtime

Informatique et mathématiques

- Les outils que nous avons vus sont basés sur des fondements mathématiques forts !
- Ils font le lien entre informatique et mathématiques
- Conseil de cours vidéos :
  `https://www.college-de-france.fr/chaire/xavier-leroy-sciences-du-logiciel-chaire-statutaire/events`

Sur la preuve de programmes avec Frama-C

- ACSL by Example
  `https://github.com/fraunhoferfokus/acsl-by-example`
- Introduction à la preuve de programmes C
  `https://allan-blanchard.fr/frama-c-wp-tutorial.html`

Deux citations de gens bons

Adoption of formal methods in various areas has dramatically improved the quality of computer systems. We anticipate that formal methods can provide similar improvement in the security of computer systems. ... Without broad use of formal methods, security will always remain fragile.

*(*Formal Methods for Security, 2016*)*

Mastering the complexity of software systems is a formidable intellectual challenge and Computer Science graduates need to understand the powerful formal methods and tools that are available. Then they can choose the right technique and tool for each task and with suitable humility deserve the title of Software Engineer.

*(*Formal Methods in Industry, 2024*)*

Contribuer à Frama-C

- Open-Source classique : remonter des bugs, aider à la doc, etc.
- Mais pas que !
  - Opportunités de stages, de thèses ...

Contribuer à Frama-C

- Open-Source classique : remonter des bugs, aider à la doc, etc.
- Mais pas que !
  - Opportunités de stages, de thèses ...

Merci de m'avoir écouté et bonne continuation !