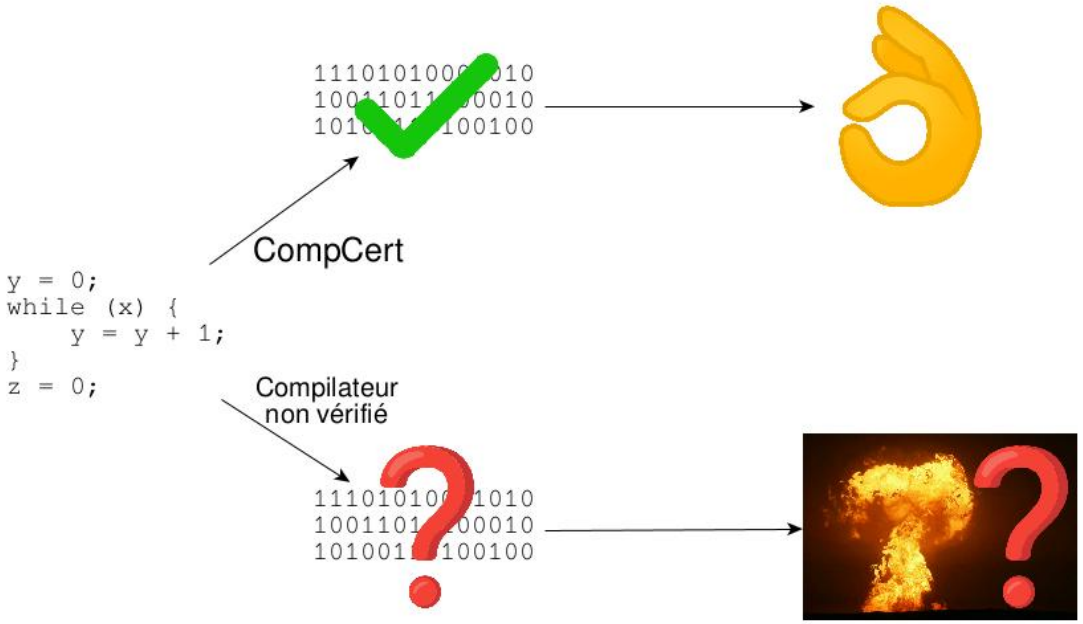
Un analyseur flots de données paramétré par un ordre d'itération pour le compilateur formellement vérifié CompCert

Roméo La Spina

Stage encadré par Sandrine Blazy et Delphine Demange (IRISA, Rennes)
M2 MPRI

14 mars - 29 juillet 2022

CompCert : un compilateur vérifié

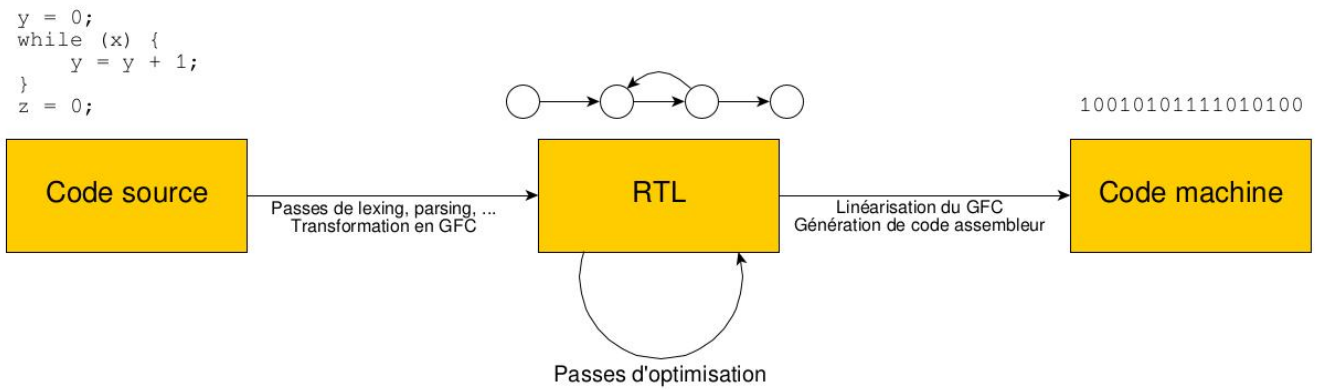


Théorème (Correction)

Tout comportement du programme cible est un comportement du programme source.

Description de CompCert (Leroy et al.)

- Compilateur C + preuve de correction en Coq
- Vérifié, mais aussi optimisant !
- 11 langages intermédiaires, dont RTL
- RTL : représentation sous forme de *graphe de flot de contrôle* (GFC)
- RTL est adapté pour faire des *analyses flots de données* $\rightarrow$ optimisations



Problématique

- Quel algorithme pour faire une analyse flots de données ?
- **Dans CompCert :** Algorithme de Kildall [Kil73]
- **But du stage :** Implémenter un autre algorithme, dû à Bourdoncle [Bou93]. Plus évolué, et utilisé en interprétation abstraite.

Analyse flots de données (1)

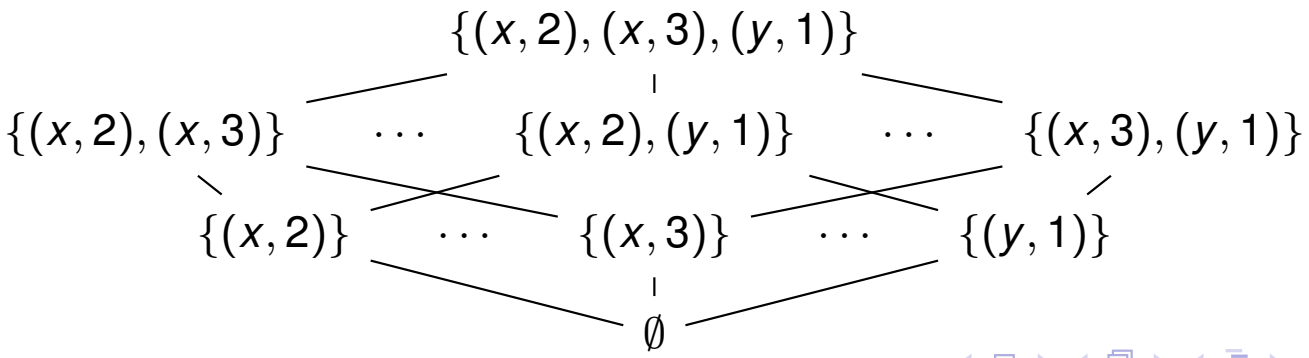
Définition (Demi-treillis)

Un *demi-treillis* est un ensemble *partiellement ordonné* $(L, \sqsubseteq)$ avec :

- un opérateur de borne supérieure $\sqcup$
- un plus petit élément $\bot$

Exemple (Demi-treillis des définitions)

Soit $\mathcal{V}$ est l'ensemble des variables, N l'ensemble des nœuds. On peut considérer le demi-treillis des définitions $(\mathcal{P}(\mathcal{V} \times N), \subseteq, \cup, \emptyset)$.



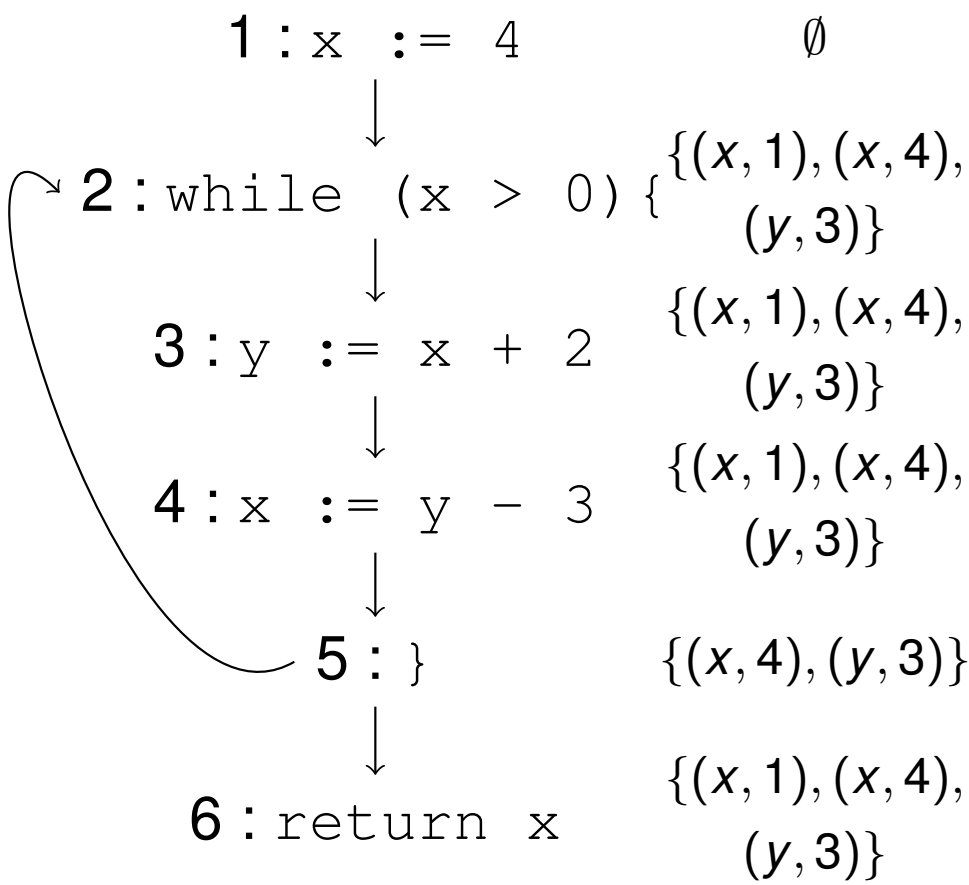
Analyse flots de données (2)

- GFC $G = (N, E)$, demi-treillis L
- Fonction de transfert

$$T : N \times L \to L$$

- **Objectif :** calculer une valeur de L à chaque nœud du graphe
- Ici, les définitions atteignantes
- Système d'équations : on cherche *aval* tel que si $n \to s$,

$$T(n, aval[n]) \sqsubseteq aval[s]$$



Algorithme de Kildall

Rappel : on cherche *aval* tel que si s est un successeur de n,

$$T(n, aval[n]) \sqsubseteq aval[s]$$

n_e est le nœud d'entrée, initialisé à v_e

```
1  W ← {n_e}
2  aval ← le mapping qui associe chaque nœud à ⊥
3  aval[n_e] ← v_e
4  tant que W ≠ ∅ faire
5  |   extraire un nœud n de W
6  |   out ← T(n, aval[n])
7  |   pour chaque successeur s de n faire
8  |   |   in ← aval[s] ⊔ out
9  |   |   si in ≠ aval[s] alors
10 |   |   |   aval[s] ← in
11 |   |   |   W ← W ∪ {s}
12 |   |   fin si
13 |   fin pour chaque
14 fin tq
15 retourner aval
```

Limitations et objectifs du stage

- Kildall : OK pour les analyses de CompCert
- *Hauteur* des demi-treillis élevée $\Rightarrow$ Temps d'analyse élevé ! !
- **Objectif :** Nouvel algorithme avec intégration facile dans CompCert (même interface, mêmes théorèmes)
- Bourdoncle donne un algorithme **paramétré par un ordre d'itération**

Etat de l'art : Solveurs vérifiés

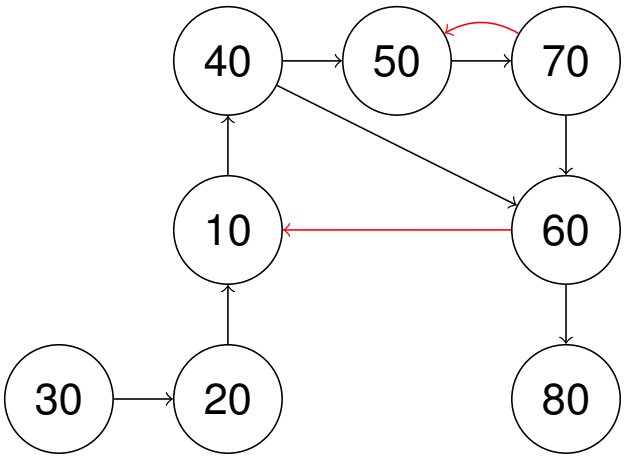
- Des solveurs vérifiés pour résoudre de tels systèmes d'équations existent... [dVPJ20, HKS10, CVH92]
- ...mais sont mal adaptés à une intégration dans CompCert
- Ces solveurs sont *locaux*, problème différent

Etat de l'art : Opérateurs d'élargissement

- Pour assurer la rapidité de convergence, opérateurs d'élargissement [CC92]
  $\Rightarrow$ Résultat moins précis
- Il faut un ensemble de nœuds où appliquer l'opérateur d'élargissement
- Bourdoncle fournit un tel ensemble $\Rightarrow$ adapté à de telles techniques
- Hors des objectifs du stage, à ajouter dans une contribution future

Tri topologique faible

- Ordre sur les nœuds du graphe + parenthésage
- **Idée :** On construit l'ordre d'itération en fonction de la structure du graphe
- **Composante :** nœuds entre deux parenthèses
- **Tête :** premier nœud d'une composante
- Pour tout arc arrière $u \to v$ de G, v est la tête d'une composante qui contient u



$$w_0 = 30\ 20\ (10\ 40\ (50\ 70)\ 60)\ 80$$

Algorithme de Bourdoncle

- **Entrées :** un GFC muni d'un t.t.f.
- **Sortie :** une solution *aval* du système d'équations
- Le t.t.f. donne l'ordre dans lequel on va traiter les nœuds
- **Stratégie itérative :** itérations sur les composantes englobantes

$$30\ 20\ [10\ 40\ 50\ 70\ 60]^*\ 80$$

- **Stratégie récursive :** itérations récursives sur les composantes

$$30\ 20\ [10\ 40\ [50\ 70]^*\ 60]^*\ 80$$

(avec $w_0 = 30\ 20\ (10\ 40\ (50\ 70)\ 60)\ 80$)

Contribution

- Intégration de la stratégie itérative de l'algorithme de Bourdoncle dans CompCert, sans élargissement
- Itération bornée pour garantir la terminaison
- Preuve de correction en Coq
- Calcul préalable du t.t.f. :
  - Méthode DFN prouvée directement en Coq
  - Méthode des sous-composantes fortement connexes, validée *a posteriori*
- Implémentation analogue à Kildall
  $\Rightarrow$ applicable à toutes les optimisations de CompCert (option `-Obourdoncle`)

Dans CompCert

- Module `Bourdoncle` : implémentation de Bourdoncle + preuve de correction
- Module `Wto` pour calculer le t.t.f.
- Le théorème final est le même, intégration facile

```
Theorem fixpoint_solution:
∀ ev res n instr s,
  (* res is the result of the analysis *)
  fixpoint ev = Some res →
  code!n = Some instr →
  (* s is a successor of n *)
  In s (successors instr) →
  (∀ n x, L.eq x L.bot → L.eq (transf n x) L.bot) →
  (* The result is a solution *)
  L.ge res!!s (transf n res!!n).
```

Idée générale de la preuve

- On maintient un ensemble S de composantes stabilisées
- **Invariant :** Pour toute composante $c \in S$, pour tout nœud n de la composante et s un de ses successeurs,

$$T(n, aval(n)) \sqsubseteq aval(s)$$

- Sachant que :
  - A la fin, toutes les composantes sont stabilisées
  - Tous les nœuds du graphe sont dans une composante

on peut conclure que le théorème de correction est vrai.

Evaluation expérimentale

- Programmes de tests fournis avec CompCert (`spass`, ...)
- Pour chaque analyse de CompCert (vivacité, code mort, ...), comparaison Kildall/Bourdoncle :
  - Temps d'analyse sans validation *a posteriori*
  - Nombre d'applications de $\sqcup$
  - Nombre d'applications de la fonction de transfert
  - Précision de l'analyse
- Utilisation de l'option `-timings`, nouvelle option `-stats`

Résultats

- Temps : Bourdoncle = 1.6-2x Kildall
- Applications de $\sqcup$: Bourdoncle = 2x Kildall
- Applications de T : Bourdoncle = 1.4x Kildall
  $\Rightarrow$ Kildall plus dynamique car pas d'ordre à respecter, mais on ne fait pas encore d'élargissement !
- Moins de 0.2% des nœuds ont une valeur abstraite différente
  $\Rightarrow$ différence de précision mineure

Conclusion et perspectives

- Première implémentation encourageante
- Ouvre la porte à des évolutions plus significatives (élargissement notamment)
- Dans le futur :
  - Stratégie récursive pour Bourdoncle
  - Preuve directe de l'algorithme des s.c.f.c.

Références I

François Bourdoncle, *Efficient chaotic iteration strategies with widenings*, Formal Methods in Programming and Their Applications (Berlin, Heidelberg) (Dines Bjørner, Manfred Broy, and Igor V. Pottosin, eds.), Springer Berlin Heidelberg, 1993, pp. 128–141.

Patrick Cousot and Radhia Cousot, *Comparing the Galois connection and widening/narrowing approaches to abstract interpretation*, Programming Language Implementation and Logic Programming (Berlin, Heidelberg) (Maurice Bruynooghe and Martin Wirsing, eds.), Springer Berlin Heidelberg, 1992, pp. 269–295.

Références II

Baudouin L Charlier and Pascal Van Hentenryck, *A universal top-down fixpoint algorithm*, Tech. report, USA, 1992.

Paulo Emílio de Vilhena, François Pottier, and Jacques-Henri Jourdan, *Spy Game : Verifying a Local Generic Solver in Iris*, Proceedings of the ACM on Programming Languages **4** (2020), no. POPL.

Martin Hofmann, Aleksandr Karbyshev, and Helmut Seidl, *Verifying a local generic solver in Coq*, Static Analysis (Berlin, Heidelberg) (Radhia Cousot and Matthieu Martel, eds.), Springer Berlin Heidelberg, 2010, pp. 340–355.

Références III

Gary A. Kildall, *A unified approach to global program optimization*, Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (New York, NY, USA), POPL '73, Association for Computing Machinery, 1973, p. 194–206.

Définitions (1)

Définition (**Tri hiérarchique**)

Un *tri hiérarchique* $\mathcal{H}$ d'un ensemble est une permutation des élements de cet ensemble. $u \preceq_{\mathcal{H}} v$ signifie "u est avant v dans le tri hiérarchique, ou $u = v$". Cette relation définit un ordre total sur les élements de l'ensemble.

Définition (**Parenthésage**)

Un *parenthésage* d'un tri hiérarchique est un parenthésage bien formé sur les éléments de la permutation donné par le tri hiérarchique, tel qu'on n'a jamais deux parenthèses ouvrantes consécutives.

Définitions (2)

Définition (**Composantes, têtes**)

Une *composante* d'un parenthésage d'un tri hiérarchique correspond aux éléments contenus entre deux parenthèses correspondantes dans le parenthésage. Le premier élement de la composante est appelé la *tête* de la composante. Pour un élément u, on note $\omega(u)$ l'ensemble des têtes des composantes qui contiennent u.

Définition (**Arc arrière**)

Etant donné un tri hiérarchique $\mathcal{H}$, un arc $(u, v) \in E$ du graphe de flot de contrôle est un *arc arrière* si $v \preceq u$.

Définitions (3)

Définition (**Tri topologique faible**)

Un *tri topologique faible* (t.t.f.) d'un graphe de flot de contrôle G est un parenthésage d'un tri hiérarchique de l'ensemble des nœuds de G, qui vérifie la propriété suivante : pour tout arc arrière $u \to v$ de G, on a $v \in \omega(u)$

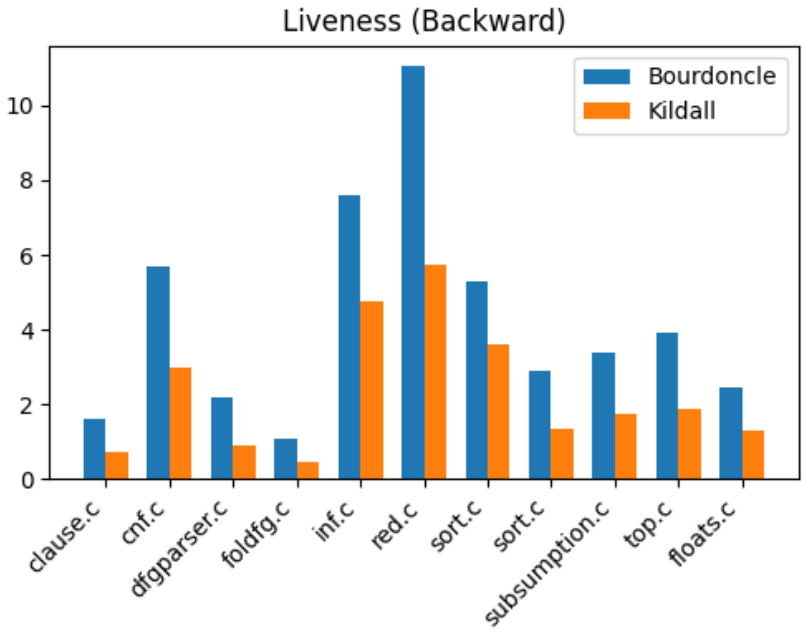
Définition (**Composantes externes**)

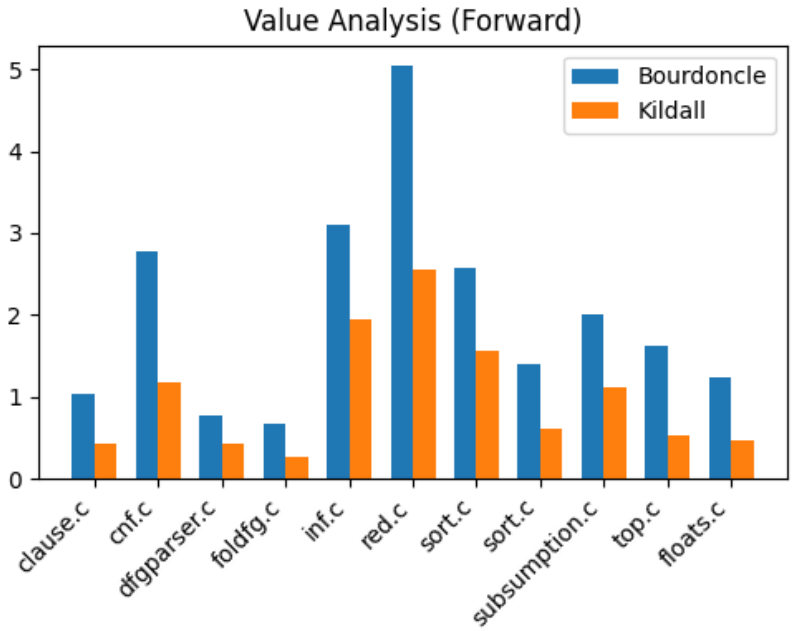
On définit $\mathcal{O}(w)$ comme étant la liste des *composantes externes* d'un t.t.f. w, c'est-à-dire la liste obtenue en appliquant H_{elt} sur les élements de w.

Algorithme de Bourdoncle, stratégie itérative

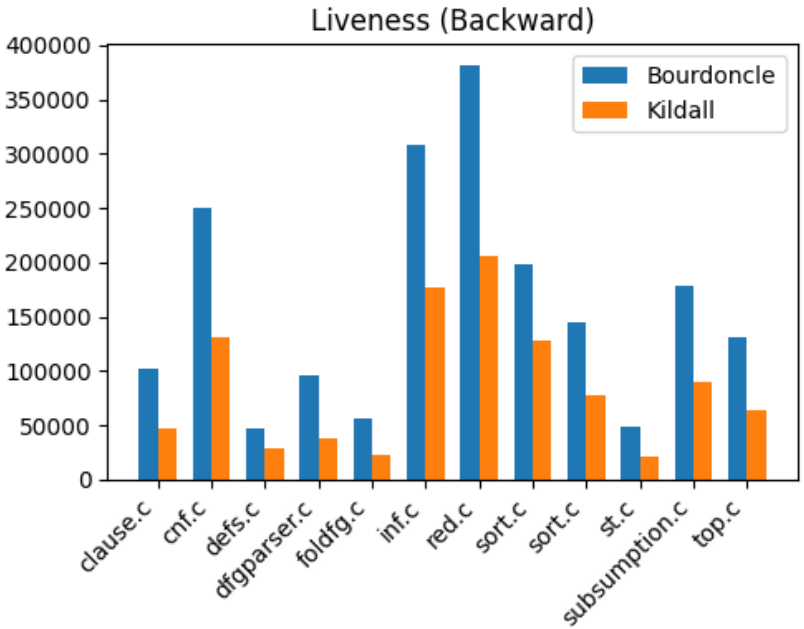
```
1  S ← ∅
2  aval ← le mapping qui associe chaque nœud à ⊥
3  aval[n_e] ← v_e
4  pour chaque c ∈ 𝒪(w) faire
5      répéter
6          m ← false
7          pour n ∈ c faire
8              out ← T(n, aval[n])
9              pour chaque s ∈ succ(n) faire
10                 in ← aval[s] ⊔ out
11                 si in ≠ aval[s] alors
12                     aval[s] ← in
13                     m ← true
14                 fin si
15             fin pour chaque
16         fin pour
17     jusqu'à m = false;
18     S ← S ∪ {c}
19 fin pour chaque
20 retourner aval
```

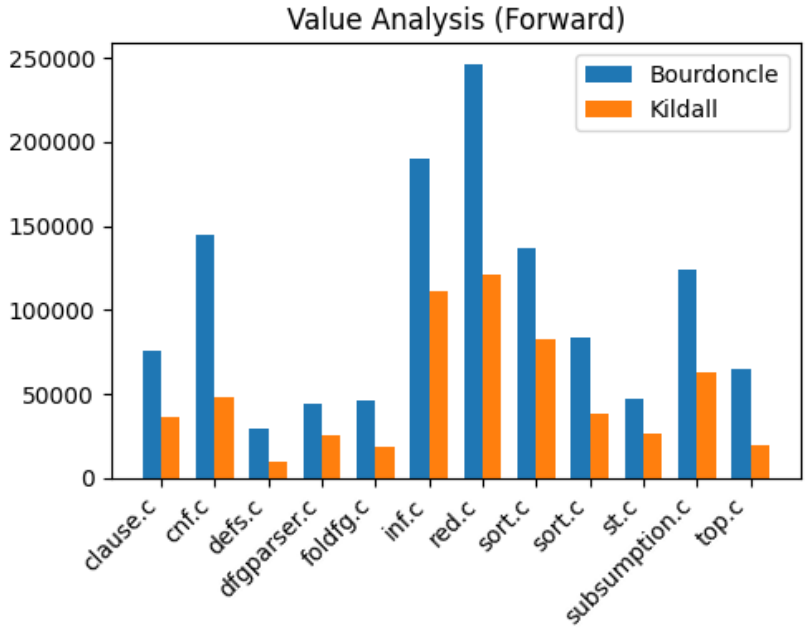
Temps d'analyse





Application de ⊔





Application de T