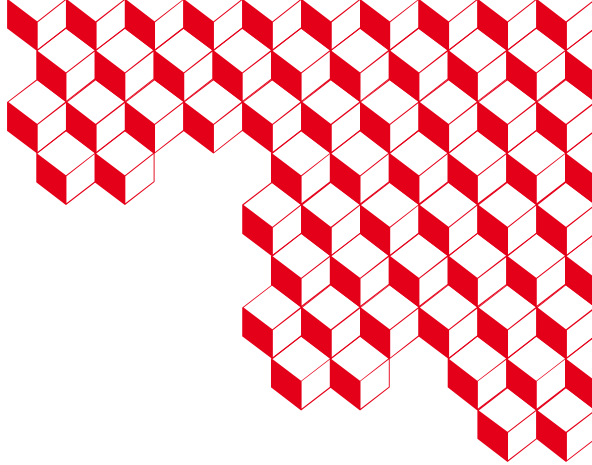


Interflop
Avancement tâche 3

Sous-tâche 3.1 – besoins en couples (front, back)
Cas d'usage

- Changement de back-end
  - Garanties supplémentaires en comparant les résultats d'une analyse – certification / qualification
  - Fournir des explications lorsqu'une analyse montre que la précision est hors clous – debug

- Changement de front-end
  - Le système sous analyse n'est pas uniforme – mix python / C++, communication par fichier
  - Besoin d'analyse de précision performante et précise sur une partie du code, mais pas partout

- Approche module par module
  - Découpage automatique et delta-debug
  - Résumés d'analyse pour conduire une nouvelle analyse

Comparaison de résultats d'analyse

- Industriel externe – application 30kloc
  - Besoin de certifier l'application auprès d'autres autorités – évaluation de la précision des résultats
  - Systèmes embarqués: courbes d'évolution de la précision au cours du temps
    - Comparaison Verrou x86-64 / verrou plate-forme embarquée
    - Comparaison Verrou – Verificarlo
    - Comparaison Verrou – Cadna : détection (Cadna) et évaluation impact (Verrou) des tests instables
      - Besoin d'évaluer pourquoi les courbes sont visuellement différentes – accumulation d'erreur ?
    - Comparaison Cadna – FLDLib : erreur conservatrice finit par diverger après quelques dizaines de cycles
- Kit de qualification d'un outil de précision – ex qualification Fluctuat (propriété CEA) - *NEW*
  - V1: comparaison exécution float et exécution réelle approchée décrit par Fluctuat (soundness)
  - A venir: intersection non vide entre les résultats de Fluctuat et de FLDLib (soundness)
  - A venir: résultats pires cas FLDLib/origins proche des bornes de Fluctuat (precision)

Expliquer les résultats d'analyse

- Cadna et FLDLib = 2 bibliothèques C++

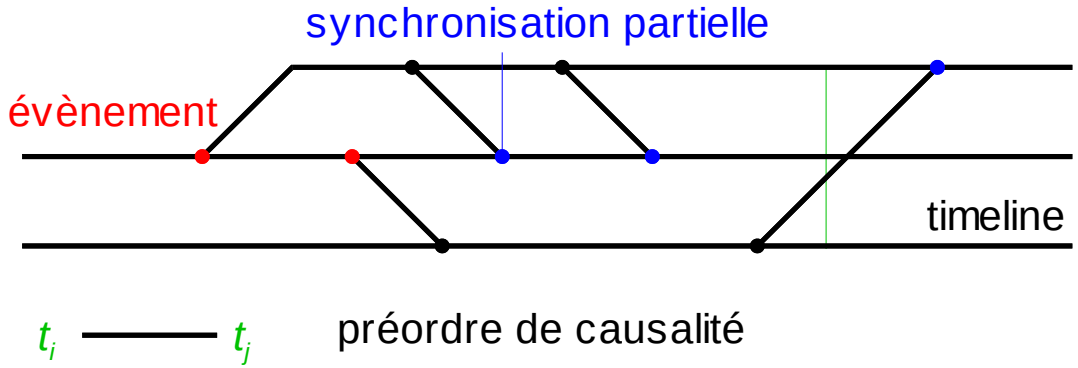
```
struct my_double {
 fld_double fld_val;
 st_double cadna_val;
};
#define double my_double
```

  Point d'arrêt du debogueur lorsque l'erreur de fld_val >> cadna_val
  Possibilité de tracer les instructions impliquées dans le calcul = divergence progressive

- Analyse conservatrice : plus l'intervalle est petit, plus les résultats sont précis - *NEW*
  - Subdiviser les domaines d'entrée et réagglomérer les résultats = plus précis qu'une analyse directe
  - Subdivisions avec exigences ⇒ isoler les cas problématiques
  - Générer un pire cas sur un cas problématique – en contrôlant l'erreur avec 54 bits de mantisse
  - Expliquer pourquoi les exigences ne sont pas tenues sur ce pire cas (exécution)

Changement de front-end / de code

- Principe général
  - Lancer 2 ou plusieurs runs d'analyse en //
    - Ex une exécution sous gdb et une analyse formelle sans main
  - Définir la politique de synchronisation
    - Si l'analyse n'a pas de valeur sur une zone mémoire, elle peut demander une valeur (et une erreur) au run arrêté
    - En fin, d'analyse un « finish » peut-être appliqué sur l'analyse et les calculs d'erreurs « reversés » au run arrêté sur le finish pour qu'il redémarre. + mettre à jour les registres



- En pratique
  - Il est plus simple d'extraire un code unitaire et des données dans un fichier,
  - de recompiler le code unitaire avec instrumentation,
  - pour analyser formellement le code et produire des données de sorties
  - qui peuvent éventuellement servir au run initial
  - pour poursuivre avec répétition des deux étapes précédentes

Approche modulaire pour les analyses conservatrices

- Analyse conservatrice : plus l'intervalle est petit, plus les résultats sont précis
- Subdivisions avec exigence = utilisation massive sur les études industrielles
- Problème 1 : les subdivisions avec exigences ne peuvent s'imbriquer
- Problème 2 : subdivision/partitionnement (énumération d'entiers) puis subdivisions avec exigences conduit à des temps d'analyse prohibitifs
- Problème 3 : les subdivisions avec exigences doivent être à un niveau de scope donné

- Solution : - *NEW*
  - trouver une formule intermédiaire qui est suffisante pour prouver les dernières exigences
  - Utiliser des subdivisions avec exigences pour prouver la formule intermédiaire
  - Refaire une analyse sans subdiviser jusqu'à la formule, puis limiter les domaines avec la formule, subdiviser avec des exigences, en limitant de nouveau les domaines subdivisés avec la formule
  - Définir plusieurs étapes intermédiaires si nécessaire

Approche modulaire pour le filtrage temporel

- Analyse conservatrice : trouver un invariant temporel
- Le back-end pour trouver l'invariant (méta back-end) ≠ back-end analyse

- Solution : - *NEW*
  - Isoler le code du filtre
  - Analyse le filtre (back-end invariant) et trouver un invariant temporel bornant le domaine et l'erreur
  - Sauvegarde de l'invariant temporel en tant qu'annotation dans le code d'origine: filtre + traitement
  - Relancer une analyse naïve sur le filtre + traitement, mais qui exploite l'invariant en sortie de filtre, pour rester précise sur le reste du code

Sous-tâche 3.2 – Mise en place d'annotations d'analyse

- Pas de solution unifiée, mais des retours d'expérience en fonction des problématiques rencontrées
- Delta-debug
- Précision Mixte
- Analyse approchée dont analyse chromatique - *NEW* - – subdivisions
- Extraction de code
- Résumés d'analyse

Conclusion

- Pas d’approche unifiée pour le moment, mais des mécanismes à avoir à l’esprit

- Combinaison d’analyse = convaincre, démontrer

- D3.1: Etat de l’art de la combinaison d’analyse à finaliser
- D3.2: Changement de front-end à finaliser