

Raisonnement relationnel sur des sémantiques monadiques

Benjamin Bonneau – Verimag

28 janvier 2025

Compilation formellement vérifiée

$\text{source} \mapsto \text{cible}$

- Sémantiques
 - Ensemble des comportements : behaves prog
- Correction
 - Raffinement :

$\text{behaves cible} \subseteq \text{behaves source}$

Sémantiques dénotationnelles

- Domaine d'objets mathématiques représentant les sémantiques
- Dénotations des éléments syntaxiques
 - Combinaison des sémantiques

$$\left[\begin{array}{l} \text{fn } f() \{ \dots \} \\ \text{fn } g() \{ \dots \} \end{array} \right]_{\text{prog}} := f \left(\begin{array}{l} \left[\text{fn } f() \{ \dots \} \right]_{\text{func}} \\ \left[\text{fn } g() \{ \dots \} \right]_{\text{func}} \end{array} \right)$$

- Sémantiques des composantes d'un programme
- Domaine indépendant de la syntaxe

- Sémantiques opérationnelles
 - CompCert (Leroy, 2009) : « petits pas », systèmes de transitions
- Vellvm (Zakowski et al., 2021) : sémantique dénotationnelles
 - Domaine : arbres d'interaction, ITrees (Xia et al., 2020)
 - Effets : divergence, non-déterminisme, évènements externes...
 - Coinductif, interface relationnelle

- Bibliothèque Rocq axiomatisant les domaines sémantiques
- Lemmes et techniques de preuves génériques

1. Domaines et opérateurs
2. Raisonnement
 - Relations et axiomes
 - Raffinements inter-procéduraux
 - Transformations de boucles
3. Un modèle : systèmes de transition étiquetés
4. Conclusion

Domaines et opérateurs

Domaine

`S (E : EventFam.t) (A : Type) : Type`

- A : type de retour
- E : évènements émis par la sémantique

`EventFam.t := Type → Type`

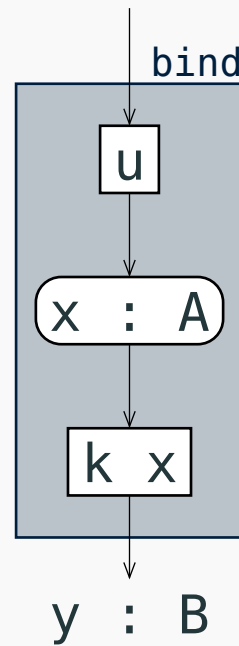
- indexés par leur type de réponse

Exemple :

```
Inductive I0 : EventFam.t :=  
  | Input                : I0 val  
  | Output (out : val) : I0 unit  
  
  S I0 (list val)
```

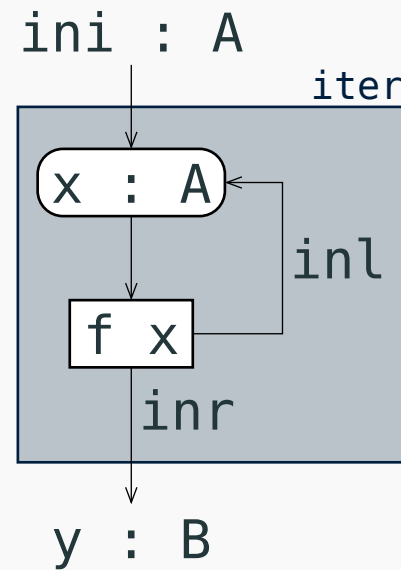
Opérateurs monadiques

- `ret [E A] (x : A) : S E A`
- `bind [E A B] (u : S E A) (k : A → S E B) : S E B`



Itération

- `iter [E A B] (f : A → S E (A + B)) (ini : A) : S E B`
 - boucles ou appels terminaux



Représentation des appels avec des événements internes (McBride, 2015)

- `trigger [E A] (e : E A) : S E A`
- `mrec [D E] (f : ∀ A (d : D A), S (D + E) A)
[Aini] (ini : D Aini) : S E Aini`

Exemple de D :

```
Inductive CallE : EventFam.t :=  
  mk_call (f : fsym) (args : list val) : CallE val
```

Raisonnement

Relations

Relation de raffinement:

$$u \gtrsim_{\text{Re}, R} v$$

- Hétérogène : $u : S \rightarrow D \rightarrow A, v : S \rightarrow E \rightarrow B$
- Post-condition : $R : A \rightarrow B \rightarrow \text{Prop}$
- Relation sur les événements : $\text{Re} : \text{EventFam}.\text{Rel} \rightarrow D \rightarrow E$

Relation d'équivalence homogène :

$$u \approx v \quad := \quad u \gtrsim_{\text{id}, \text{id}} v \wedge v \gtrsim_{\text{id}, \text{id}} u$$

Axiomes

- Propriétés de $\approx$

$$\frac{U \approx_{\text{Re0}, R0} V \quad V \approx_{\text{Re1}, R1} W}{U \approx_{\text{Re0}; \text{Re1}, R0; R1} W} \text{TRANS}$$

- Morphismes

$$\frac{U \approx_{\text{Re}, r} V \quad \forall x y. r \ x \ y \Rightarrow f \ x \approx_{\text{Re}, R} g \ y}{\text{bind } U \ f \approx_{\text{Re}, R} \text{bind } V \ g} \text{REFBIND}$$

- Équivalences

$$\frac{}{\text{bind } (\text{ret } x) \ k \approx k \ x} \text{RETL}$$

$$\frac{}{\text{iter } f \ x \approx r \leftarrow f \ x; \begin{cases} \text{iter } f \ x' & \text{si } r = \text{inl } x' \\ \text{ret } y & \text{si } r = \text{inr } y \end{cases}} \text{ITERUNFOLD}$$

Relations sur les évènements

```
EventFam.Rel (D E : EventFam.t) : Type :=  
  ∀ [A B] (d : E A) (e : D B)  
    (post : A → B → Prop), Prop
```

Raffinement des évènements :

$$\frac{\text{Re } d \text{ e } R}{\text{trigger } d \approx_{\text{Re}, R} \text{trigger } e} \text{REFTRIGGER}$$

Définition de la relation identité sur les évènements :

$$\frac{e : E \ A}{\text{id}_E \ e \ e \ \text{id}_A}$$

Transformations inter-procédurales

Inductive `CallE : EventFam.t :=`
 `mk_call (f : fsym) (args : list val) : CallE val`

- Pre et post conditions

$$\frac{\text{pre } f \text{ args}}{\text{Re}_{\text{spec}} (\text{mk_call } f \text{ args}) (\text{mk_call } f \text{ args}) (\lambda x \ y. \ x=y \wedge \text{post } f \ x)}$$

- Spécialisation

$$\frac{}{\text{Re}_{\text{spe}} (\text{mk_call } f \text{ args}) (\text{mk_call } f_0 \text{ args}) \text{id}_{\text{val}}}$$

$$\frac{\text{pre args}}{\text{Re}_{\text{spe}} (\text{mk_call } f \text{ args}) (\text{mk_call } f_1 \text{ args}) \text{id}_{\text{val}}}$$

Transformations de boucles

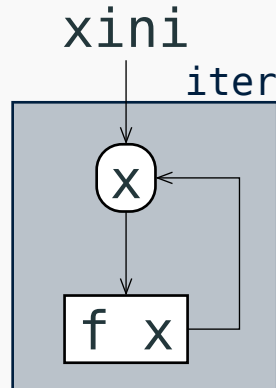
`loop1 := while True: f     $\approx$     while True: { f; f } =: loop2`

$\llbracket \text{loop1} \rrbracket = \text{iter } (\lambda x0. x1 \leftarrow f \ x0; \text{ret } (\text{inl } x1)) \ xini$

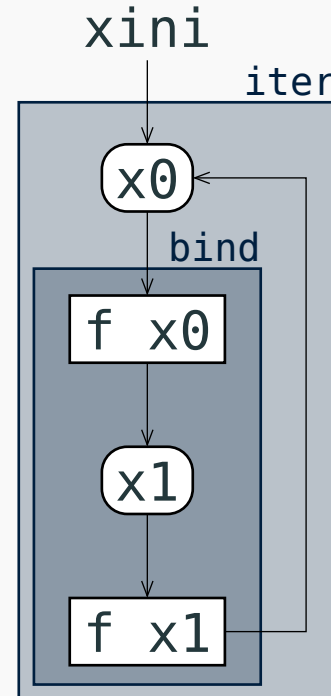
$\approx$

$\llbracket \text{loop2} \rrbracket = \text{iter } (\lambda x0. x1 \leftarrow f \ x0; x2 \leftarrow f \ x1; \text{ret } (\text{inl } x2)) \ xini$

loop1



loop2



Transformations de boucles

- Expression de bind avec iter :

```
loop2  =  while True: { f; f }  
      ≈  while True: for i in [0, 1]: f
```

- Fusion des itérateurs imbriqués (codiagonale) :

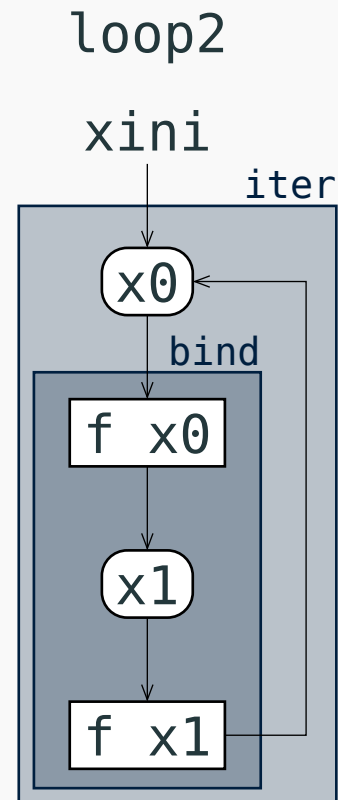
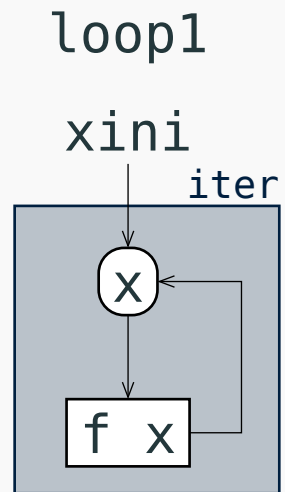
```
      ≈  for i in [0,1,0,1,...]: f
```

- Réindexage :

```
      ≈  while True: f  =  loop1
```

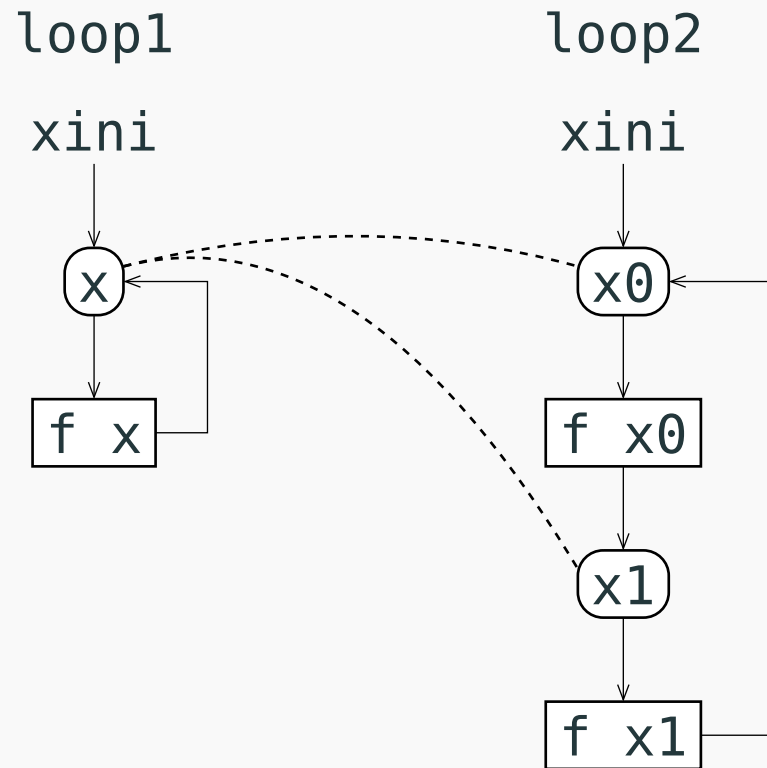
Intuition graphique

Graphe de flot de contrôle



Intuition graphique

Graphe de flot de contrôle



- Morphisme de CFG (Gourdin et al., 2023)

Inlining

- Vus comme des appels terminaux:

```
let rec loop1 () =  
    f; loop1 ()
```

$\approx$

```
let rec loop2 () =  
    f; f; loop2 ()
```

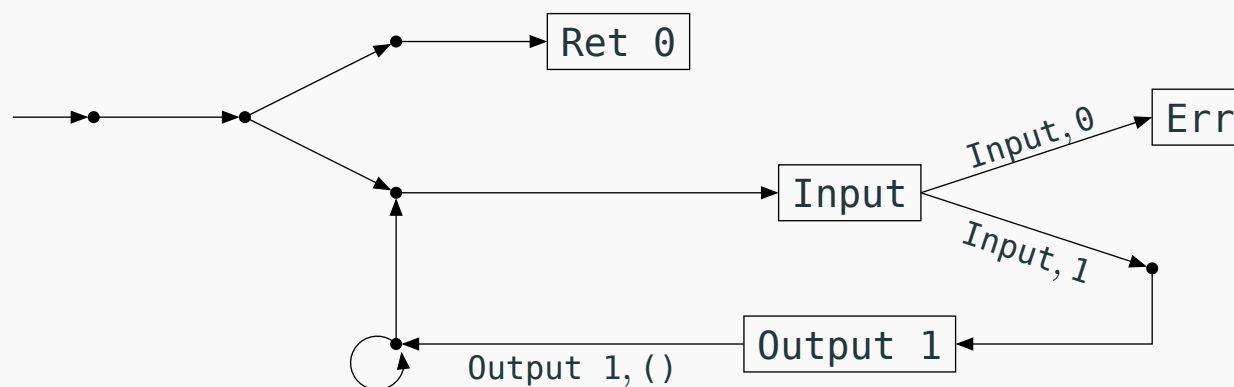
- Inlining pour la récursion générale :
 - Même technique appliquée à mrec

Un modèle : systèmes de transition étiquetés

$$S := LTS$$

Effets: divergence, non-déterminisme, comportements indéfinis

Exemple d'un système de transitions LTS IO val :



- ... états $\xrightarrow{\text{évènement}}$ transitions

Opérateurs

- `ret x`

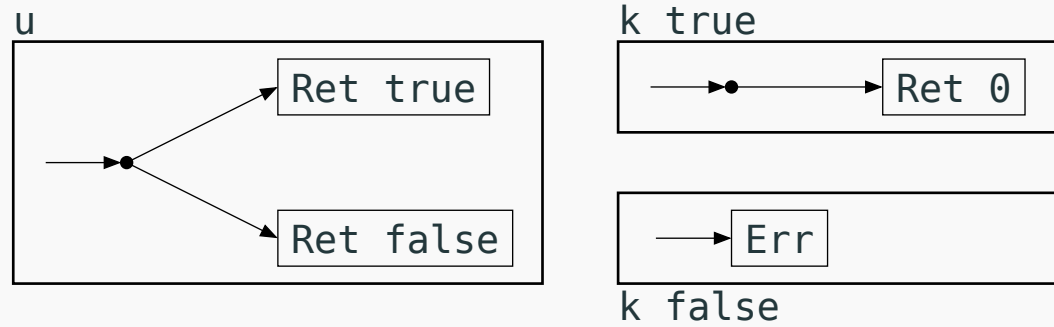
→ Ret x

Opérateurs

- `ret x`



- `bind u k`

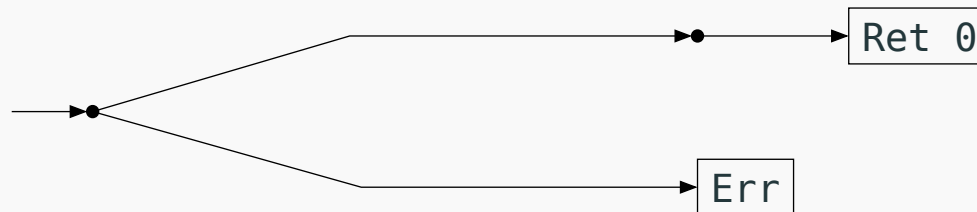


Opérateurs

- `ret x`

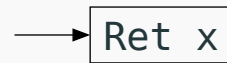


- `bind u k`

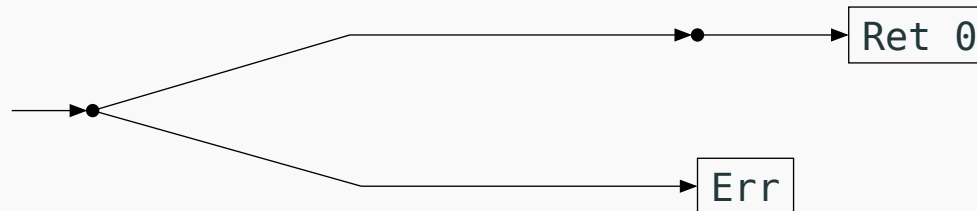


Opérateurs

- `ret x`



- `bind u k`



- `mrec` : pile d'appel

- *Backward simulation* modulo les séquences finies de pas silencieux
- Correct mais pas complet par rapport à l'inclusion des comportements
- Preuves des axiomes : coinduction

Conclusion

- Domaines expressifs pour les sémantiques dénotationnelles
 - ITrees, LTS
- Théorie générique : vis-à-vis du langage et du domaine
- Permet d'abstraire les effets et de renforcer la modularité
 - Exemple: élimination de code mort (DCE), cf. article

Bibliographie

- Gourdin, L., Bonneau, B., Boulmé, S., Monniaux, D., & Bérard, A. (2023). Formally Verifying Optimizations with Block Simulations. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2), 59-88. <https://doi.org/10.1145/3622799>
- Leroy, X. (2009). A Formally Verified Compiler Back-end. *Journal of Automated Reasoning*, 43(4), 363-446. <https://doi.org/10.1007/s10817-009-9155-4>
- McBride, C. (2015). Turing-Completeness Totally Free. In *Mathematics of Program Construction: Vol. 9129. Mathematics of*

Program Construction (p. 257-275). Springer International Publishing. https://doi.org/10.1007/978-3-319-19797-5_13

Xia, L.-y., Zakowski, Y., He, P., Hur, C.-K., Malecha, G., Pierce, B. C., & Zdancewic, S. (2020). Interaction trees: representing recursive and impure programs in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL), 1-32. <https://doi.org/10.1145/3371119>

Zakowski, Y., Beck, C., Yoon, I., Zaichuk, I., Zaliva, V., & Zdancewic, S. (2021). Modular, Compositional, and Executable Formal Semantics for LLVM IR. *Proc. ACM Program. Lang.*, 5(ICFP). <https://doi.org/10.1145/3473572>