Spécifications de programmes réactifs

PAUL Nicolas

nicolas.paul1@etu.univ-orleans.fr

17 juillet 2025

Plan

Programmation réactive

- Deux catégories de systèmes : transformationnels et réactionnels.
- Contraintes temporelles.
- Paradigme dédié pour la programmation réactive synchrone ou asynchrone.
- Plusieurs implantations en utilisation :
  - Lustre (86) dans Ansys SCADE ; ou
  - Rx.js (18) sur le Web.

Vérification formelle

- Importante mais souvent ignorée.
- Trois étapes :
  1. spécification,
  2. synthétisation, et
  3. satisfaction.
- Des méthodes existent pour automatiser : BNF, MATLAB/Simulink, Astrée, &c.

Types et logiques, OCamlFRP

- Lien entre logiques et types ?
- Oui, la correspondance Hurry-Coward.
- Le typage, familier aux ingénieurs.
- Typage faible, algébrique et dépendant.
- Exemple avec Ada chez Thalès et Lockheed Martin.

```
Value
get(size_t i)
{
    return array[i];
}
```

- OoB si $i >= n$ ou $i < 0$.
- Contrôle manuel de la validité au call-site.

Types et logiques, OCamlFRP

```
fn get() -> Option<Value> {
    if i < 0 || i >= array.len() {
        unsafe { Some(array.get_unchecked(i)) }
    } else {
        None
    }
}
```

- OoB sous contrôle et explicites.
- Mais verbreux à chaque call-site.

```
type Index is range 1 .. 5;
type Bounded_Array is array (Index) of Value;
function Get(i: Index) returns Value
is
   My_Array: Bounded_Array = [V1, V2, V3, V4, V5];
begin
   return My_Array (i);
end Get;
```

- OoB impossible : le code ne compilera pas si on appelle Get avec une valeur possiblement nulle.
- Le call-site reste simple.

Types et logiques, OCamlFRP

Qu'est-ce que la FRP ?

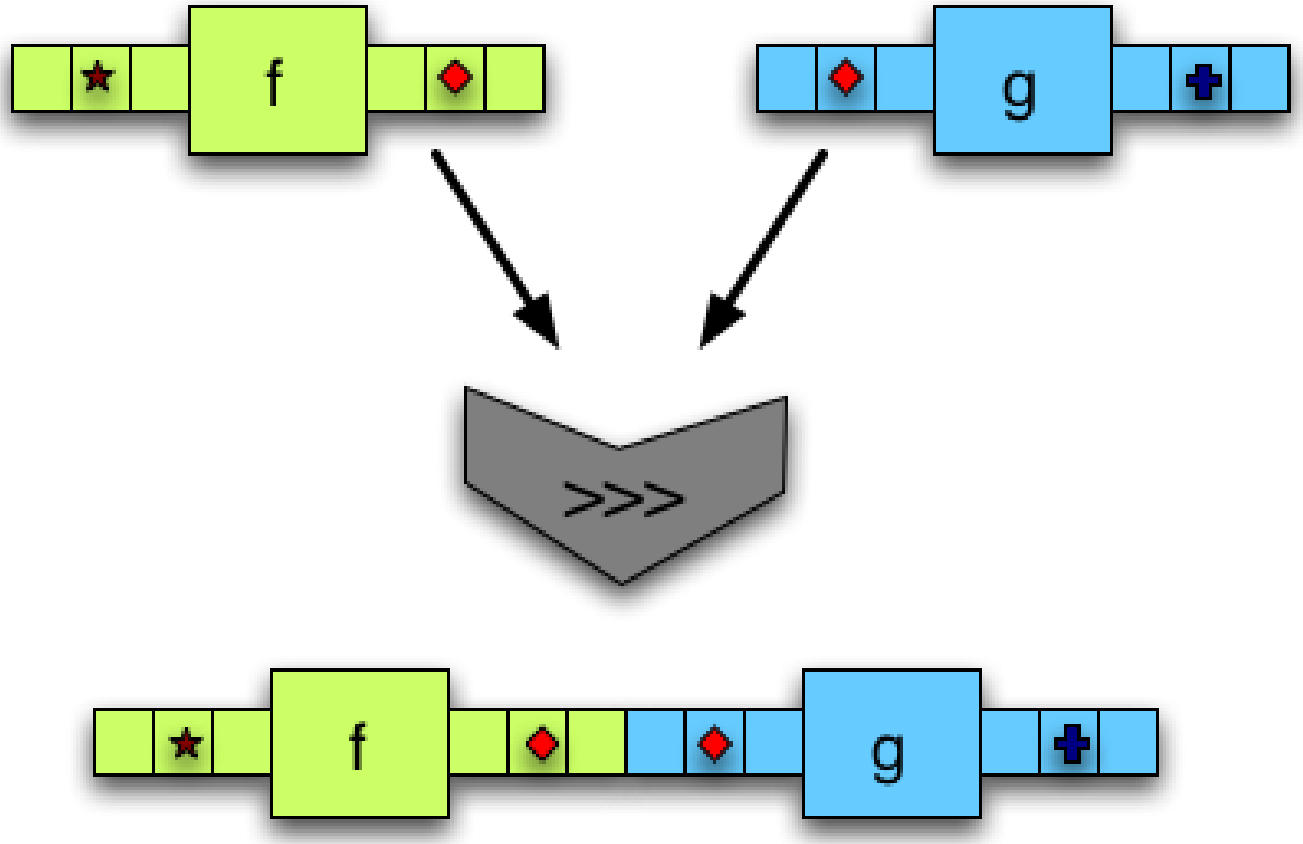
- Paradigme déclaratif.
- Fran (89) pour remplacer DirectX Animations.
- $\mathit{Behavior}_\alpha = \mathit{Time} \longrightarrow \alpha$
- $\mathit{Event}_\alpha = \mathit{Time} \times \alpha$
- Tutoriel officiel de Fran.
- Mais fuites mémoires et temporelles.

Types et logiques, OCamlFRP

Une alternative, OCamlFRP :

- Fuites réglées par la co-itération.
- $Stream = (S \longrightarrow T \times S) \times S$
- Arrows de HUGHES John (00).
- Expressif, concis, système de type puissant.

Types et logiques, OCamlFRP



(Source : Haskell Wiki)

Types et logiques, OCamlFRP

Démonstration d'un jeu Snake.

```
let score =
    arr (func s -> { s with score = s+1 })
(* ... *)

let snake = calculate_head >>> move_head
let hit = new_apple >>> score
let apple = collision >>> choice hit move_tail
let render = fanin id id >>> fanout make_scene id

let game = loop (snake >>> apple >>> render) s0
```

Vérification déductive, Hardy

- Déduire des lemmes depuis la spécification et la synthèse.
- Si toutes les lemmes sont prouvés alors la synthèse satisfait la spécification.
- Moins de unit-tests et proche du code final.

Vérification déductive, Hardy

D'où Hardy :

- Langage impératif pour les programmes réactifs synchrones.
- Linear Temporal Logic : $P|\neg\phi|\phi\vee\psi|X\phi|\phi U\psi|\phi R\psi|G\phi|F\phi$.
- Safety and liveness properties.
- Automates de Büchi et triplets de Hoare.
- Compilation et intégration avec Why3.

Vérification déductive, Hardy

Démonstration d'une télécommande TV.

```
OFF:
| button_power {
  // Power button pressed, turn the TV on!
  vol  := 20;
  chan := "1";
  emit true to powered;
  emit vol to volume;
  emit chan to channel;
} => ON
```

Conclusion

- Augmentation du nombre de systèmes réactifs dans le futur.
- Le paradigme de programmation réactive se popularise dans des domaines non-critiques.
- La vérification formelle reste rare en dehors de quelques industries.
- Mais certaines méthodes sont de plus en plus adoptées ! Rust ?
- Simplifier l'expérience utilisateur peut aider.