Audition pour le poste de Maître de Conférence

Adrien Durier

23 mai 2023

Université Paris-Saclay - Laboratoire des Méthodes Formelles

Parcours de Recherche

- **Concurrence** : Thèse ENS Lyon – Bologne **2016 – 2019**
- **Compilation Sécurisée** : INRIA – Inst. Max Planck **2019 – 2021**
- **Systèmes Cyber-Physiques** : IRT SystemX – Saclay **2022 – 2023**

Parcours de Recherche

- **Concurrence** : Thèse ENS Lyon – Bologne **2016 – 2019**
  - Nouvelle *technique de preuve* coinductive [1]
  - Résolution d'un problème ouvert par Robin Milner [2]
- **Compilation Sécurisée** : INRIA – Inst. Max Planck **2019 – 2021**
- **Systèmes Cyber-Physiques** : IRT SystemX – Saclay **2022 – 2023**

---

[1] *Divergence and unique solution of equations* : CONCUR, LMCS
Towards 'up to context' reasoning about higher-order processes : TCS
Coq 1700 lignes

[2] *Eager functions as processes* : LICS, TCS

Parcours de Recherche

- **Concurrence** : Thèse ENS Lyon – Bologne **2016 – 2019**
  - Nouvelle *technique de preuve* coinductive [1]
  - Résolution d'un problème ouvert par Robin Milner [2]
- **Compilation Sécurisée** : INRIA – Inst. Max Planck **2019 – 2021**
  - Technique coinductive pour la cybersécurité [3]
  - Compilateur compartementalisé avec partage mémoire [4]
- **Systèmes Cyber-Physiques** : IRT SystemX – Saclay **2022 – 2023**

---

[3] *Trace-Relating Compiler Correctness...* : ESOP, TOPLAS
Coq 3000 lignes

[4] *SecurePtrs : Proving Secure Compilation...* : CSF
Coq Projet 32000 lignes *total* + Backend 1700 lignes

Parcours de Recherche

- **Concurrence** : Thèse ENS Lyon – Bologne **2016 – 2019**
  - Nouvelle *technique de preuve* coinductive [1]
  - Résolution d'un problème ouvert par Robin Milner [2]
- **Compilation Sécurisée** : INRIA – Inst. Max Planck **2019 – 2021**
  - Technique coinductive pour la cybersécurité [3]
  - Compilateur compartementalisé avec partage mémoire [4]
- **Systèmes Cyber-Physiques** : IRT SystemX – Saclay **2022 – 2023**
  - Sûreté de fonctionnement du *véhicule autonome*
  - Réalisation d'un modèle pour systèmes cyber-physiques [5]

---

[5] Rapport technique 3SA (*Interne, IRT SystemX*)
Librairie HOL-CyberPhy (**Isabelle/HOL**, 3000 lignes)
Projet de dépôt open-source sur *Archive of Formal Proofs*

Parcours de Recherche

- **Concurrence** : Thèse ENS Lyon – Bologne **2016 – 2019**
  - Nouvelle *technique de preuve* coinductive [1]
  - Résolution d'un problème ouvert par Robin Milner [2]
- **Compilation Sécurisée** : INRIA – Inst. Max Planck **2019 – 2021**
  - Technique coinductive pour la cybersécurité [3]
  - Compilateur compartementalisé avec partage mémoire [4]
- **Systèmes Cyber-Physiques** : IRT SystemX – Saclay **2022 – 2023**
  - Sûreté de fonctionnement du *véhicule autonome*
  - Réalisation d'un modèle pour systèmes cyber-physiques [5]

- **4 publications en conférences** (LICS, CONCUR, ESOP, CSF)
- **4 publications dans des journaux** (LMCS, TCS, TOPLAS)
- **1 rapport technique** (IRT SystemX)

Deux traditions de la concurrence

Tony Hoare (*Prix turing 1980*)

1978 : CSP

Communicating Sequential Processes

Robin Milner (*Prix turing 1991*)

1980 : CCS

Calculus of Communicating Systems

Deux traditions de la concurrence

Tony Hoare (*Prix turing 1980*)

1978 : CSP

Communicating Sequential Processes

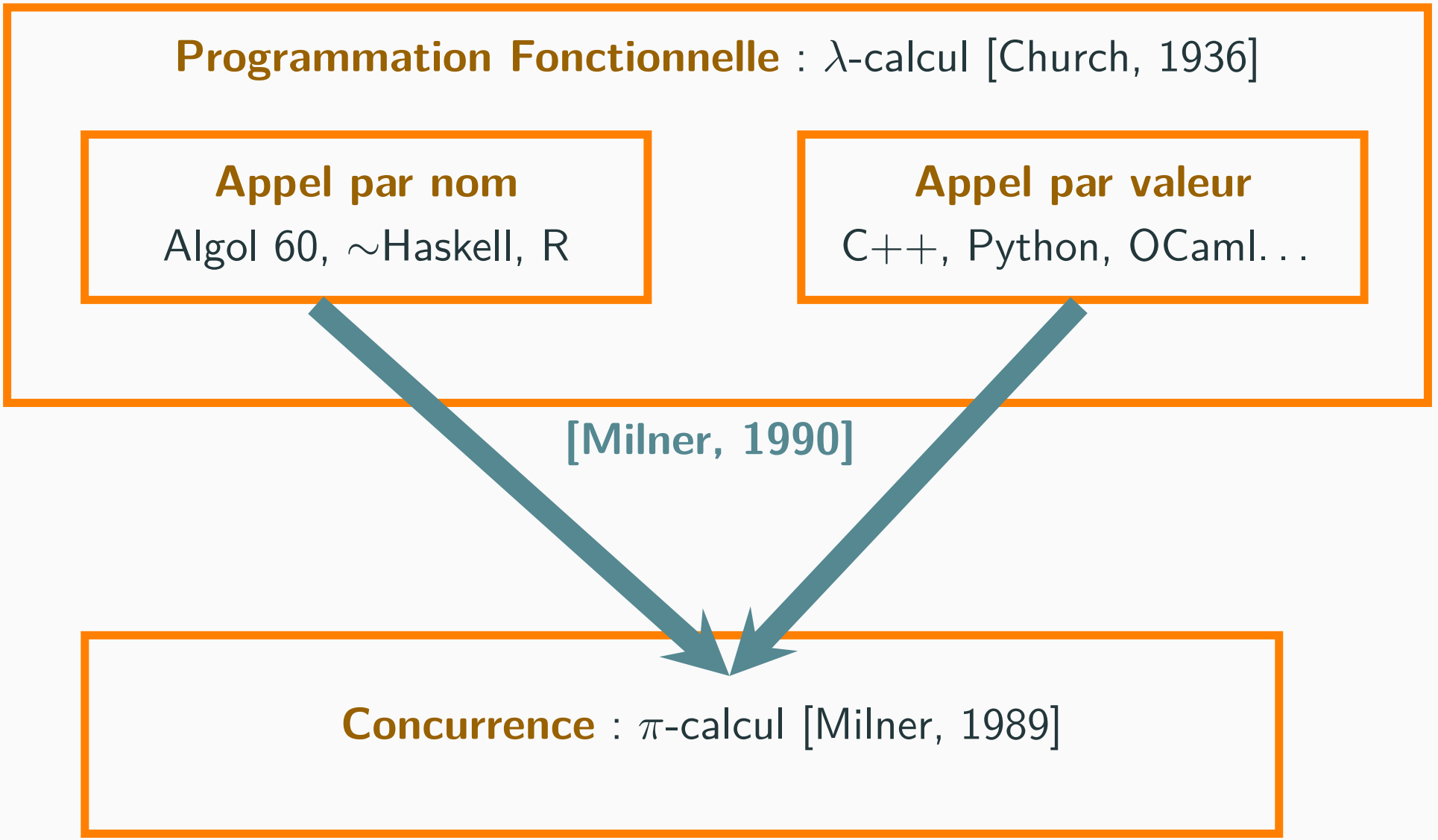
Modèles dénotationnel,
Outils de Model-Checking

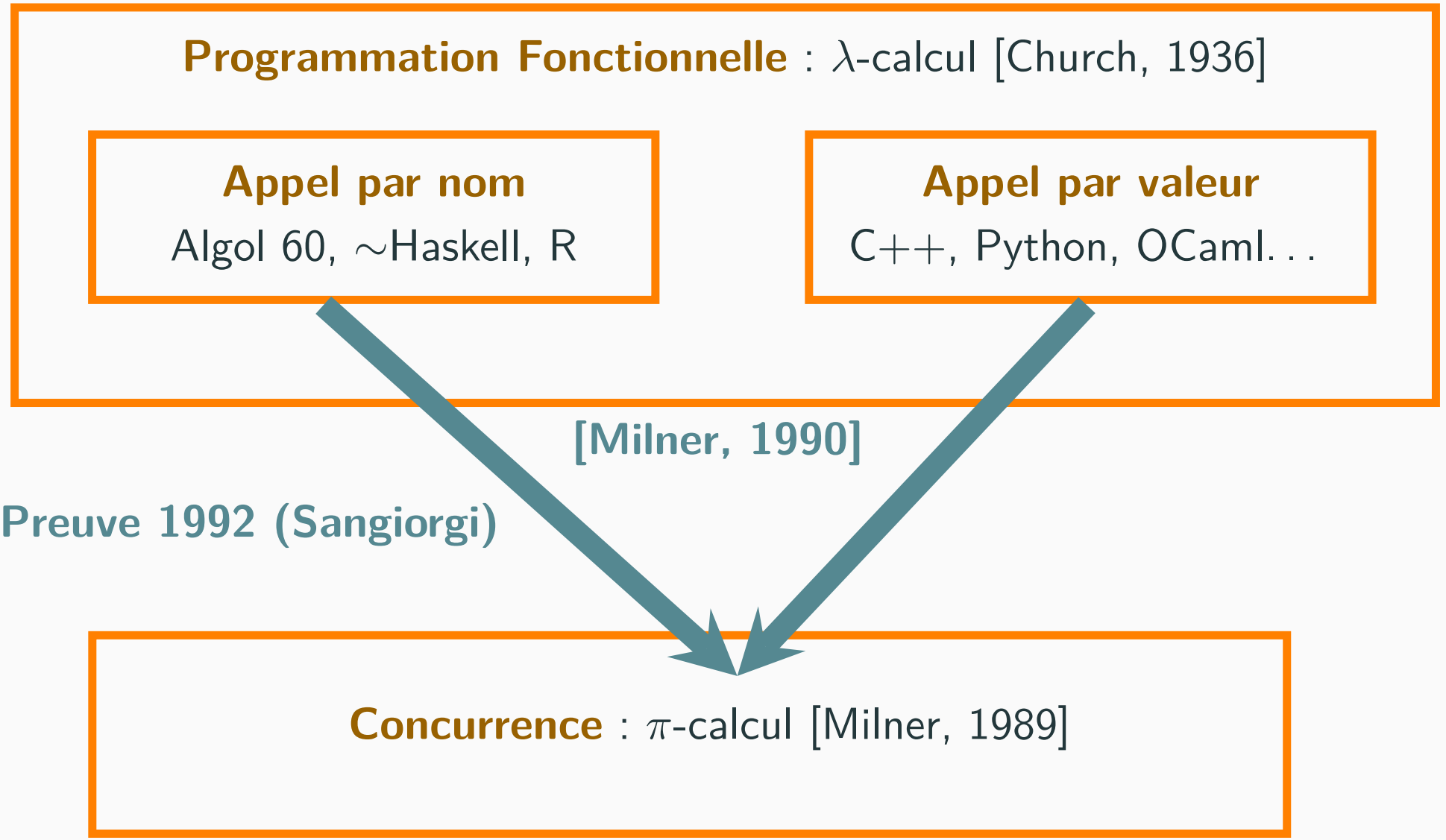
Robin Milner (*Prix turing 1991*)

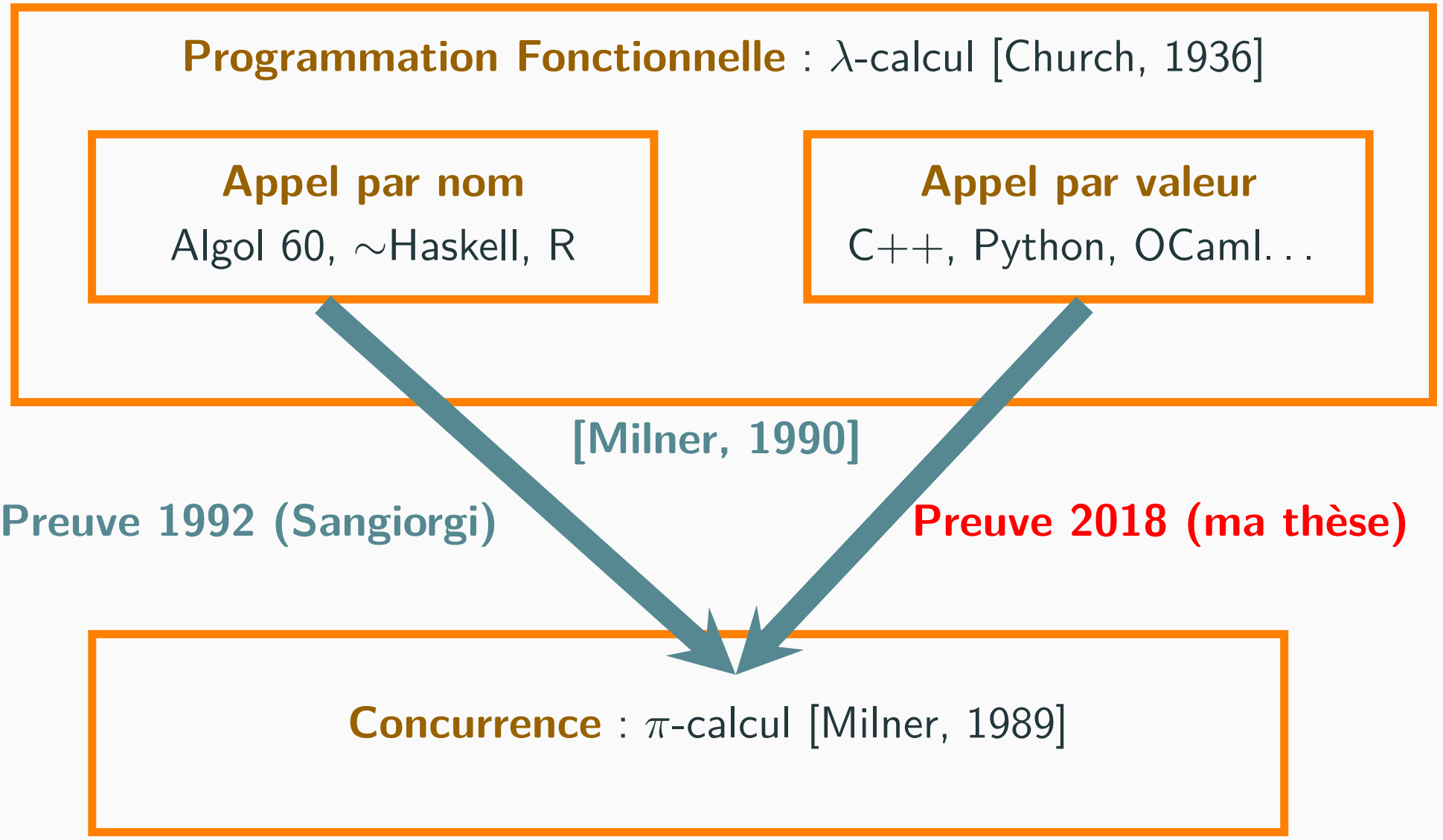
1980 : CCS

Calculus of Communicating Systems

π-calcul (1989)
communication + sophistiquée

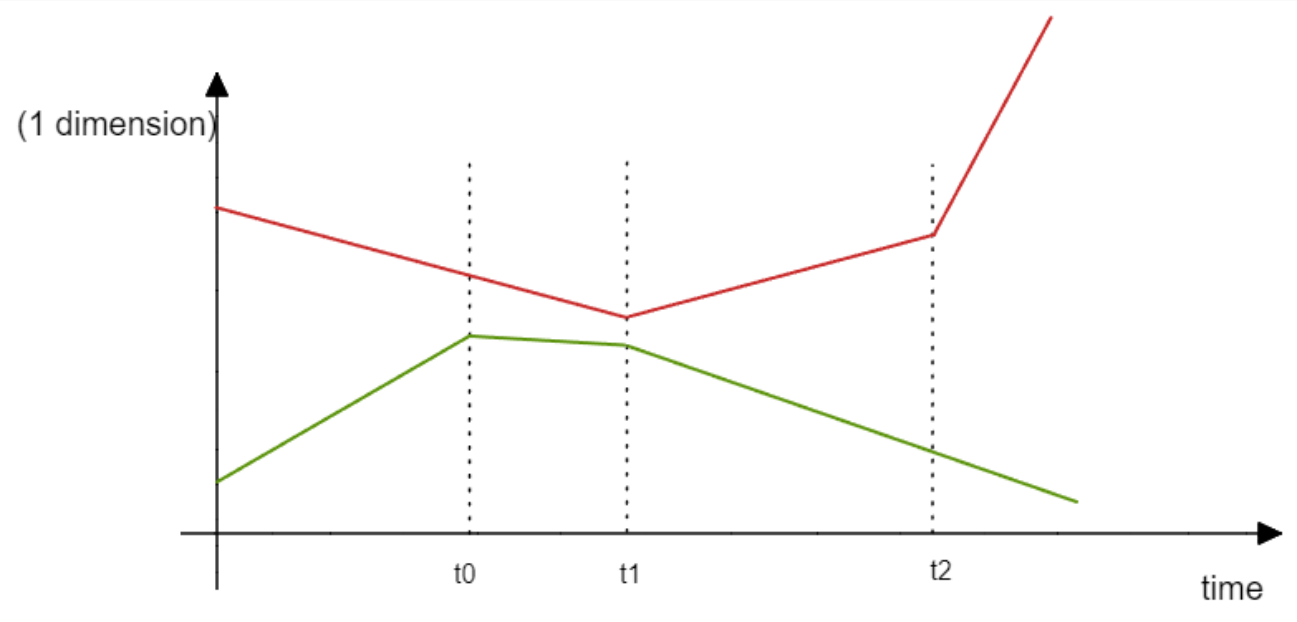

Programmation Fonctionnelle : λ-calcul [Church, 1936]
Appel par nom
Algol 60, ∼Haskell, R
Appel par valeur
C++, Python, OCaml...
[Milner, 1990]
Concurrence : π-calcul [Milner, 1989]


Programmation Fonctionnelle : λ-calcul [Church, 1936]
Appel par nom
Algol 60, ∼Haskell, R
Appel par valeur
C++, Python, OCaml...
[Milner, 1990]
Preuve 1992 (Sangiorgi)
Concurrence : π-calcul [Milner, 1989]


Programmation Fonctionnelle : λ-calcul [Church, 1936]
Appel par nom
Algol 60, ∼Haskell, R
Appel par valeur
C++, Python, OCaml...
[Milner, 1990]
Preuve 1992 (Sangiorgi)
Preuve 2018 (ma thèse)
Concurrence : π-calcul [Milner, 1989]

Post-doctorat (LMF/IRT SystemX) : HOL-CyberPhy

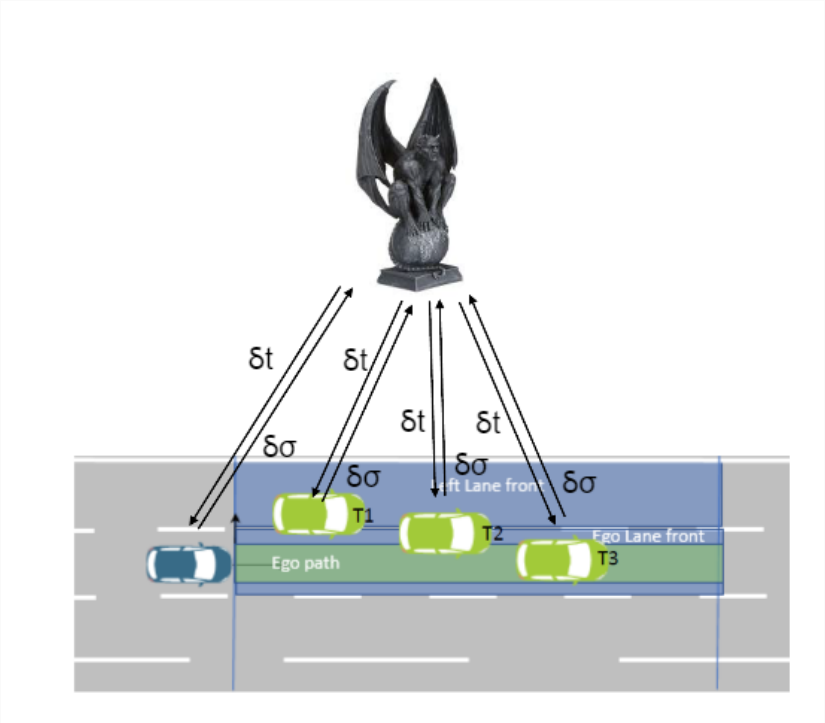
Modèle **discret** + **continu**



- Notre libraire **HOL-CyberPhy** utilise **CSP**
- **CSP** représente la composante **discrète** ('cyber') du modèle.

Post-doctorat (LMF/IRT SystemX) : HOL-CyberPhy

- Discrétisation du temps par un **programmes CSP**
- Les `agents` sont des **programmes CSP**



- La sûreté de fonctionnement est représentée par un raffinement de programmes CSP.

Projet de Recherche

Deux Axes de Recherche

Intégration au pôle **Preuves et Langages**

Appliqué : Sûreté des Systèmes Cyber-Physiques

- Partenaires industriels : Renault, Stellantis, Valeo, Airbus...
- Équipes :
  - *Preuves de programmes*
  - Liens thématiques pôle Modèles (*Systèmes critiques*)

Théorique : Programmation Fonctionnelle et Concurrence

- Équipes :
  - *Fondements du calcul, langages et compilation*
  - Liens thématiques pôle Modèles (*Concurrence et distribué*)

1er Axe : Systèmes Cyber-Physiques

- **Intégration**
  - Collaboration avec Burkhart Wolff (Preuves de programmes)
  - Équipe Systèmes critiques
- **Transfert technologique**
  - A travers la collaboration avec l'IRT SystemX (projet CVH)
  - Renault, Stellantis, Valeo
  - Validation et standardisation (SOTIF)

Roadmap

Pour homologuer et certifier : exigences **réelles** et **industrielles**.

- **Objectif à court terme** :
  1. Équations différentielles et cinématiques arbitraires
  2. Facteurs externes (vent...)
  3. Composition de stratégies de conduite

- **Objectif à long terme** :
  1. Dangers de sortie de route
  2. Génération de plans de test, génération de code
  3. **Exigences industrielles pour le plan de conception**

2e Axe : Appel Par Valeur et Concurrence

Encodages :

- ☑ Appel par nom
  - ☑ Équivalences de programmes
- ☑ Appel par valeur (**langages de programmation réels** : OCaml...)

2e Axe : Appel Par Valeur et Concurrence

Encodages :

- ☑ Appel par nom
  - ☑ Équivalences de programmes
- ☑ Appel par valeur (**langages de programmation réels** : OCaml...)
  - ☐ Équivalences de programmes

2e Axe : Appel Par Valeur et Concurrence

Encodages :

- ☑ Appel par nom
  - ☑ Équivalences de programmes
- ☑ Appel par valeur (**langages de programmation réels** : OCaml...)
  - ☐ Équivalences de programmes
  - ☐ Réduction forte
- ☐ Appel par nécessité

2e Axe : Appel Par Valeur et Concurrence

Encodages :

- ☑ Appel par nom
  - ☑ Équivalences de programmes
- ☑ Appel par valeur (**langages de programmation réels** : OCaml...)
  - ☐ Équivalences de programmes
  - ☐ Réduction forte
- ☐ Appel par nécessité

Collaboration possible avec *Thibault Balabonski*

Enseignement

Enseignement à l'ENS et à l'Université (Lyon)

ENS Lyon (L3) : 176h

- Théorie de la programmation (3 ans) 3x32h
- Projet de Programmation (1 an) 24h
- Algorithmique (1 an) 32h

Université Lyon 1 (L2) :

- TP Algorithmique et programmation avancée (1 an) 16h

---

192 heures équivalents TD

Création de sujets :

- Exercices TD
- Sujets TP (fiches OCaml, Coq...)
- Devoirs à la maison et corrections

Implication dans l'enseignement

Autres activités d'enseignement (ENS Lyon) :

- Jury de soutenance de stages (L3, M1) — **eq. TD : 6h**
- Organisation de cours de soutien
  - Soutien général / Remise à niveau L3 — **eq. TD : 10h**
  - Par matière, à la demande — **+h**

Mediation et engagement :

- Encadrement du stage de programmation pour lycéennes **Girls Can Code !**
- Engagement dans le soutien scolaire en mathématiques (collège, lycée)

Projet d'Enseignement

Projets Hors Recherche

- **POPLmark Challenge** : Terminaison du Système F — Coq
- **etherweb.fr** : Création et de gestion de sites web — PHP/JS
- **Algorithme de Buchberger** : Bases de Gröbner — OCaml
- **Compilation** : Pascal $\mapsto$ SPIM — OCaml
- **TL2** : Mémoire partagée efficace — Java
- **latexifier** : décompilation PDF vers LaTeX — C++/Web

Projet d'enseignement

Besoins principaux :

- Cours appliqués en Licence
- Responsabilités administratives

Cours :

Génie Logiciel	L2/L3 (Avancé)/M1 MIAGE
Système	L2 (Architecture)/L3 (OS)
Web	L3/M1 (Sécurité)
Programmation	fonction., objet, C, projets – L1/L2/L3
Algorithmique	L1/L2/L3/M

$\longrightarrow$ Prise en charge **dès septembre**

Nouveaux Cours Proposés

Dans un ou deux ans : cours à l'interface entre applications et méthodes formelles :

1. **Projet de Compilateur** (Programmation)
   - Niveau Master
   - Possibilité d'adapter le langage aux besoins du département
2. **Compilation Sécurisée**
   - Sujet moderne, enjeux importants (navigateurs)
   - Appliqué, mais nécessite des propriétés mathématiques
   - Pas de preuve formelle
3. **Concurrence et coinduction**
   - Informatique fondamentale

Questions

Merci pour votre attention !

Appendices

1. Conclusion/Résumé 1
2. Publications 2
3. Equivalences de programmes 3
4. Encodage de Milner 4
5. Call-by-name vs call-by-value 5
6. Compilation sécurisée 6
7. Cyber-Physical Systems 7

Conclusion

- Thèse à l'ENS Lyon et l'Università di Bologna
- Post-doc à l'INRIA et l'institut Max Planck
- Post-doc au LMF

Enseignement

- 192 heures en ACE
- Principalement envers des élèves d'une ENS
- Investissement pédagogique

Recherche

- Calculs de processus, sémantique
  Assistants de preuves (Coq, désormais Isabelle)
  Compilation sécurisée
- 5 articles, 4 dans des conférences de rang A ou A+, 4 dans des journaux réputés

Parcours

2014 : **M1** Informatique fondamentale, ENS Lyon

2015 : **M2** Recherche MPRI

2016-2020 : **Doctorat** en cotutelle LIP, ENS Lyon et Università di Bologna, sous la direction de Daniel Hirschkoff et Davide Sangiorgi

2019-2021 : **Post-doc** avec Catalin Hritcu (INRIA puis Institut Max Planck, Bochum)

2022 - : **Post-doc** joint IRT SystemX/LMF, avec Paolo Crisafulli, Safouan Taha, Burkhart Wolff

Publications

2017 *Divergence and unique solution of equations*
CONCUR 2017, LMCS 2019

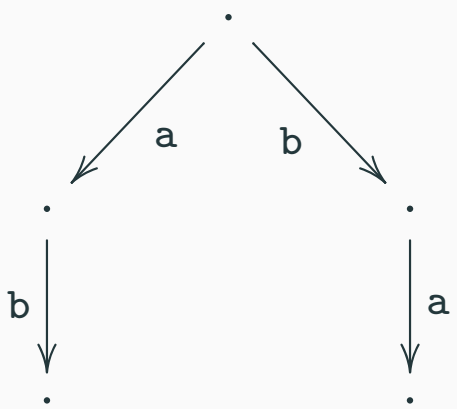
2018 *Eager Functions as Processes*
LICS 2018, Theoretical Computer Science 2022

2020 *Towards 'up to context' reasoning about HO processes*
Theoretical Computer Science 2020

2020 *Trace-Relating Compiler Correctness and Secure Compilation*
ESOP 2020, TOPLAS 2021

2022 *SecurePtrs : Proving Secure Compilation with Data-Flow Back-Translation and Turn-Taking Simulation*
CSF 2022

Concurrence



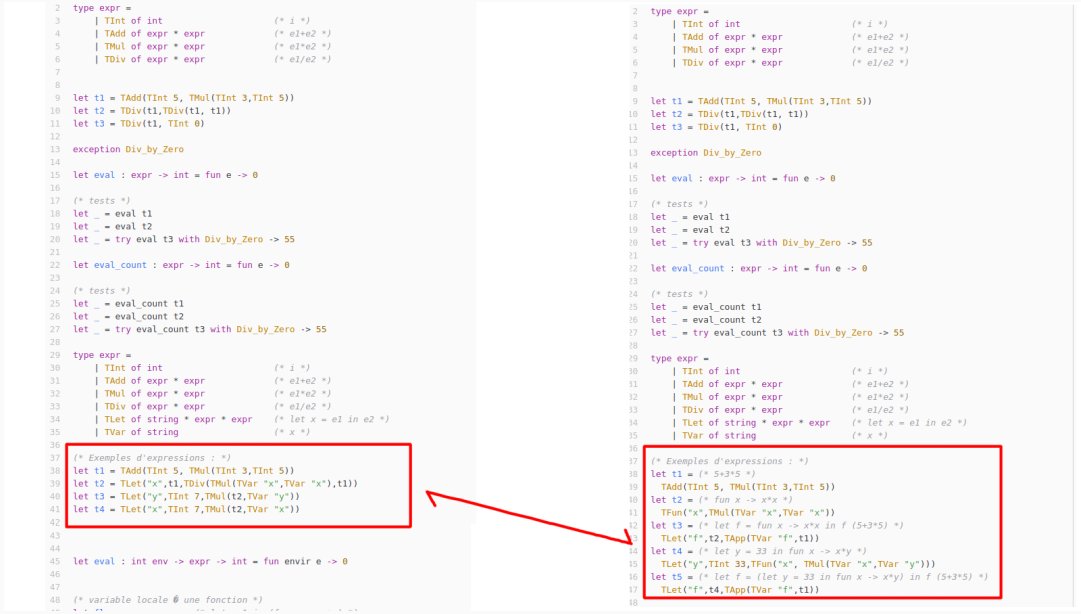
→ a et b peuvent avoir lieu dans n'importe quel ordre

→ Pas de lien de **causalité**

→ Ce sont des évènements **concurrents**

Equivalence de Programme

Contexte : Étude formelle des languages de programmation



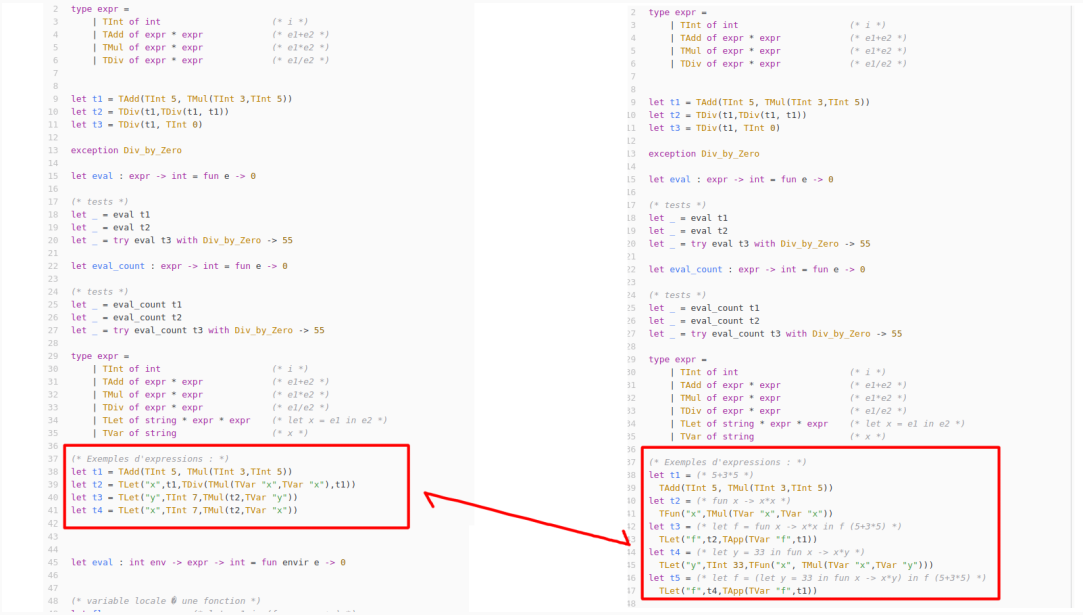
```
type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)


let t1 = TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = TDiv(t1,TDiv(t1, t1))
let t3 = TDiv(t1, TInt 0)

exception Div_by_Zero

let eval : expr -> int = fun e -> 0

(* tests *)
let _ = eval t1
let _ = eval t2
let _ = try eval t3 with Div_by_Zero -> 55

let eval_count : expr -> int = fun e -> 0

(* tests *)
let _ = eval_count t1
let _ = eval_count t2
let _ = try eval_count t3 with Div_by_Zero -> 55

type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)
    | TLet of string * expr * expr  (* let x = e1 in e2 *)
    | TVar of string                (* x *)

(* Exemples d'expressions : *)
let t1 = TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = TLet("x",t1,TDiv(TMul(TVar "x",TVar "x"),t1))
let t3 = TLet("y",TInt 7,TMul(t2,TVar "y"))
let t4 = TLet("x",TInt 7,TMul(t2,TVar "x"))


let eval : int env -> expr -> int = fun envir e -> 0


(* variable locale à une fonction *)
```

```
type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)


let t1 = TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = TDiv(t1,TDiv(t1, t1))
let t3 = TDiv(t1, TInt 0)

exception Div_by_Zero

let eval : expr -> int = fun e -> 0

(* tests *)
let _ = eval t1
let _ = eval t2
let _ = try eval t3 with Div_by_Zero -> 55

let eval_count : expr -> int = fun e -> 0

(* tests *)
let _ = eval_count t1
let _ = eval_count t2
let _ = try eval_count t3 with Div_by_Zero -> 55

type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)
    | TLet of string * expr * expr  (* let x = e1 in e2 *)
    | TVar of string                (* x *)

(* Exemples d'expressions : *)
let t1 = (* 5+3*5 *)
  TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = (* fun x -> x*x *)
  TFun("x",TMul(TVar "x",TVar "x"))
let t3 = (* let f = fun x -> x*x in f (5+3*5) *)
  TLet("f",t2,TApp(TVar "f",t1))
let t4 = (* let y = 33 in fun x -> x*y *)
  TLet("y",TInt 33,TFun("x", TMul(TVar "x",TVar "y")))
let t5 = (* let f = (let y = 33 in fun x -> x*y) in f (5+3*5) *)
  TLet("f",t4,TApp(TVar "f",t1))
```

Objectifs

- **Vérifier** une optimisation
- **Prouver** qu'on peut remplacer un programme par un autre

Equivalence de Programmes



```
type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)


let t1 = TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = TDiv(t1,TDiv(t1, t1))
let t3 = TDiv(t1, TInt 0)

exception Div_by_Zero

let eval : expr -> int = fun e -> 0

(* tests *)
let _ = eval t1
let _ = eval t2
let _ = try eval t3 with Div_by_Zero -> 55

let eval_count : expr -> int = fun e -> 0

(* tests *)
let _ = eval_count t1
let _ = eval_count t2
let _ = try eval_count t3 with Div_by_Zero -> 55

type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)
    | TLet of string * expr * expr  (* let x = e1 in e2 *)
    | TVar of string                (* x *)

(* Exemples d'expressions : *)
let t1 = TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = TLet("x",t1,TDiv(TMul(TVar "x",TVar "x"),t1))
let t3 = TLet("y",TInt 7,TMul(t2,TVar "y"))
let t4 = TLet("x",TInt 7,TMul(t2,TVar "x"))


let eval : int env -> expr -> int = fun envir e -> 0


(* variable locale à une fonction *)
```

```
type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)


let t1 = TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = TDiv(t1,TDiv(t1, t1))
let t3 = TDiv(t1, TInt 0)

exception Div_by_Zero

let eval : expr -> int = fun e -> 0

(* tests *)
let _ = eval t1
let _ = eval t2
let _ = try eval t3 with Div_by_Zero -> 55

let eval_count : expr -> int = fun e -> 0

(* tests *)
let _ = eval_count t1
let _ = eval_count t2
let _ = try eval_count t3 with Div_by_Zero -> 55

type expr =
    | TInt of int                   (* i *)
    | TAdd of expr * expr           (* e1+e2 *)
    | TMul of expr * expr           (* e1*e2 *)
    | TDiv of expr * expr           (* e1/e2 *)
    | TLet of string * expr * expr  (* let x = e1 in e2 *)
    | TVar of string                (* x *)

(* Exemples d'expressions : *)
let t1 = (* 5+3*5 *)
  TAdd(TInt 5, TMul(TInt 3,TInt 5))
let t2 = (* fun x -> x*x *)
  TFun("x",TMul(TVar "x",TVar "x"))
let t3 = (* let f = fun x -> x*x in f (5+3*5) *)
  TLet("f",t2,TApp(TVar "f",t1))
let t4 = (* let y = 33 in fun x -> x*y *)
  TLet("y",TInt 33,TFun("x", TMul(TVar "x",TVar "y")))
let t5 = (* let f = (let y = 33 in fun x -> x*y) in f (5+3*5) *)
  TLet("f",t4,TApp(TVar "f",t1))
```

- Programmes partiels : x peut être indéfini
- Indistinguable dans tout contexte
  $\rightarrow$ on peut modifier le reste du code
- **Contexte :** Programme avec un trou, qui peut être remplacé par un programme partiel.

Equivalence de Programmes

Contextes

Si le contexte C est :

```
int x;
[·]
x = x++;
```

Si le programme (partiel) **P** est

```
.
x = 3;
.
```

Alors $C[\mathbf{P}]$ est le programme :

```
int x;
x = 3;
x = x++;
```

L'encodage de Milner en appel par valeur

une exécution est modélisé comme une **interaction** entre

le programme
(en train d'être évalué)

le contexte
(dans lequel il est évalué)

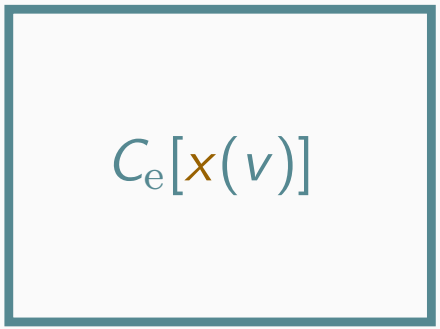
Les valeurs sont stockées par le contexte

Le programme les demande lorsqu'ils deviennent nécessaires

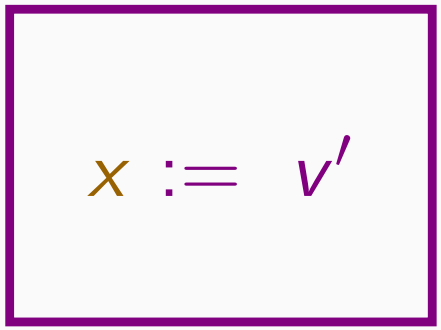
L'encodage de Milner en appel par valeur

une exécution est modélisé comme une **interaction** entre

le programme



le contexte



L'encodage de Milner en appel par valeur

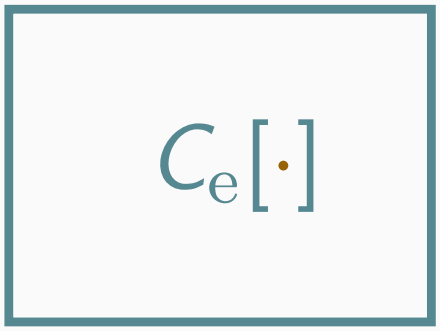
une exécution est modélisé comme une **interaction** entre

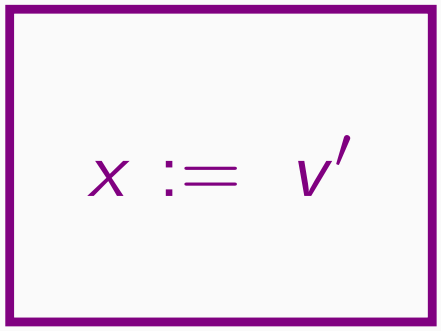
le programme **le contexte**

L'encodage de Milner en appel par valeur

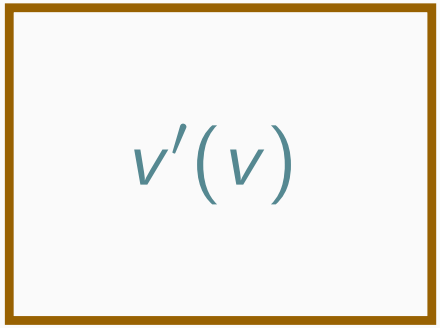
une exécution est modélisé comme une **interaction** entre

le programme **le contexte**





l'évaluation de $x(v)$ débute dans un nouvel emplacement



L'encodage de Milner en appel par valeur

une exécution est modélisé comme une **interaction** entre

le programme $C_e[y]$

le contexte $x := v'$

une fois évalué, il est stocké dans une nouvelle variable y

$y := v''$

L'appel par nom et par valeur

On étudie le problème d'**Abstraction Complète** :

Caractériser l'**équivalence $\approx$ induite** par l'encodage :

$$M \approx N \text{ iff } [\![M]\!] \simeq [\![N]\!]$$

(Pour comparer M et N, je compare leurs encodages)

L'appel par nom et par valeur

On étudie le problème d'**Abstraction Complète** :
Caractériser l'**équivalence $\approx$ induite** par l'encodage :
$M \approx N$ iff $[\![M]\!] \simeq [\![N]\!]$
(Pour comparer M et N, je compare leurs encodages)

En ***appel par nom*** :

- Résolu [Sangiorgi92]
- Correspond aux **arbres de Lévy-Longo** (Équivalence classique)
- La preuve repose sur les **techniques modulo**

L'appel par nom et par valeur

On étudie le problème d'**Abstraction Complète** :
Caractériser l'**équivalence $\approx$ induite** par l'encodage :
$M \approx N$ iff $[\![M]\!] \simeq [\![N]\!]$
(Pour comparer M et N, je compare leurs encodages)

En ***appel par nom*** :

- Résolu [Sangiorgi92]
- Correspond aux **arbres de Lévy-Longo** (Équivalence classique)
- La preuve repose sur les **techniques modulo**
  $\rightarrow$ **sensible à "l'efficacité"**

En ***appel par valeur*** :

- *Durier,Hirschkoff,Sangiorgi 2018*
- Les équivalences ne sont pas aussi bien étudiées/comprises
- Introduit de l'**inefficacité**
  $\rightarrow$ **échec des techniques modulo**

efficacité : nombre de réductions

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

Complétude de l'encodage

Théorème

Définition : Unicité des solutions

$$X \simeq f(X)$$

f vérifie **l'unicité des solutions** si :

Quand $\mathbf{P} \simeq f(\mathbf{P})$ **et** $\mathbf{Q} \simeq f(\mathbf{Q})$

Alors

$\mathbf{P} \simeq \mathbf{Q}$

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

1. Construire un système d'équation
2. Prouver que $[\![M]\!]$ et $[\![N]\!]$ sont solutions
3. Prouver que le système a une unique solution

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

1. Construire un système d'équation
2. Prouver que $[\![M]\!]$ et $[\![N]\!]$ sont solutions
3. Prouver que le système a une unique solution

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

1. Construire un système d'équation
2. Prouver que $[\![M]\!]$ et $[\![N]\!]$ sont solutions
3. Prouver que le système a une unique solution

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

1. Construire un système d'équation
2. Prouver que $[\![M]\!]$ et $[\![N]\!]$ sont solutions
3. Prouver que le système a une unique solution

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

1. Construire un système d'équation
2. Prouver que $[\![M]\!]$ et $[\![N]\!]$ sont solutions
3. Prouver que le système a une unique solution

Complétude de l'encodage

Théorème

$$M \simeq_{e\eta} N \quad \text{si et seulement si} \quad [\![M]\!] \simeq [\![N]\!]$$

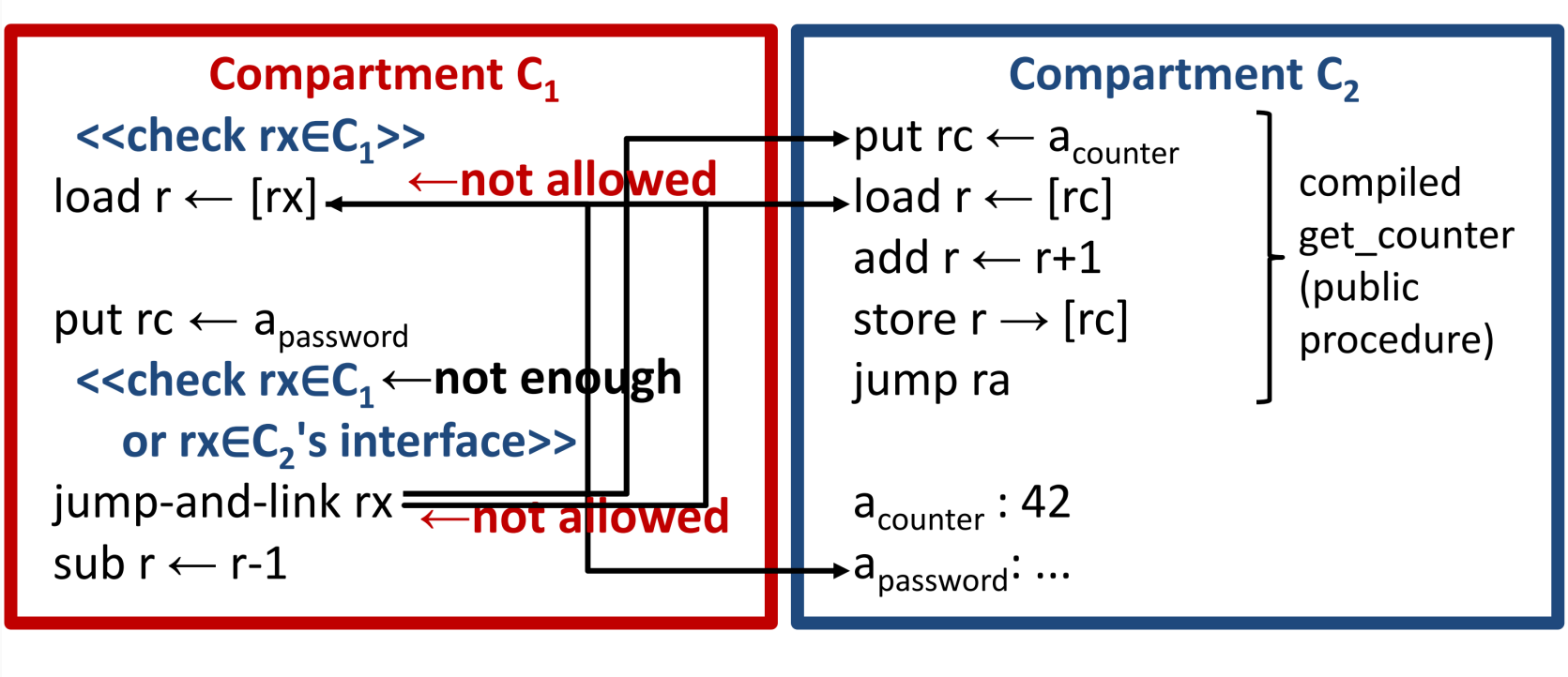
Lemme (Complétude de l'encodage)

$$\text{Si } M \simeq_{e\eta} N \quad \text{alors} \quad [\![M]\!] \simeq [\![N]\!]$$

1. Construire un système d'équation
2. Prouver que $[\![M]\!]$ et $[\![N]\!]$ sont solutions
3. Prouver que le système a une unique solution
   - $\rightarrow$ Nouvelle technique de preuve
   - $\rightarrow$ Fondée sur les théories existantes d'unicité des solutions
     Hoare[1978], Milner [1980], Roscoe [1982], Sangiorgi [2015]...

Liens avec la Compilation Sécurisée

- Sécurité par **compartementalisation** :



→ **Interactions** entre modules : programme et contexte

→ Preuve de sécurité par techniques coinductives (**CompCert**)

Trace-Relating Compiler Correctness and Secure Compilation (2020)

Contexte : Correction d'un compilateur par preuve de simulation.

Toute trace produite par le programme compilé

peut être produit par le programme source

Compilation correcte :

$$\forall \mathbf{P}.\forall t.\ \mathbf{P} \downarrow\rightsquigarrow t \Rightarrow \mathbf{P} \rightsquigarrow t$$

CompCert : La preuve de correction inverse une preuve de simulation entre la source et la **cible**.

Trace-Relating Compiler Correctness and Secure Compilation (2020)

Relations de traces : Si la **cible** et la source produisent des traces différentes ; pour prendre en compte les comportements indéfinis (C), etc.

Un exemple simple : Compilation d'un langage avec booléens et entiers vers un langage avec **seulement des entiers**
Relation sur les entiers (étendue aux traces) :

$$n \sim \mathbf{n} \qquad \mathit{true} \sim \mathbf{n} \text{ si } \mathbf{n} > 0 \qquad \mathit{false} \sim \mathbf{0}$$

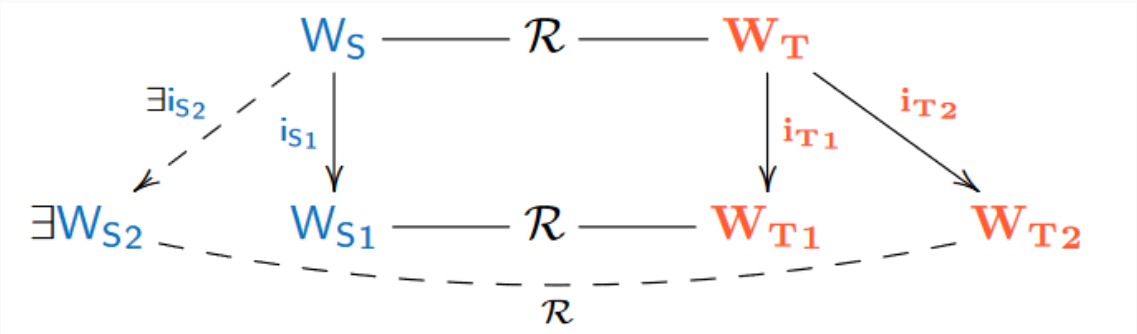
Compilation correcte :

$$\forall \mathbf{P}.\forall \mathbf{t}.\ \mathbf{P} \downarrow\!\rightsquigarrow \mathbf{t} \Rightarrow \exists t \sim \mathbf{t}.\mathbf{P} \rightsquigarrow t$$

Trace-Relating Compiler Correctness and Secure Compilation (2020)

$\rightarrow$ la technique d'inversion de simulation ne fonctionne plus si la relation n'est pas injective

$\rightarrow$ il faut une condition supplémentaire (prouvée en coq, avec une nouvelle technique de simulation faible)



Application à la compilation sécurisée :

Introduire des contextes arbitraires dans la propriété de correction (propriétés de compilation sécurisées RTP, RTC...)

Safety Properties

Example : for autonomous vehicules, $P = \texttt{no}_{\texttt{collision}}$

Safety Properties

Example : for autonomous vehicules, $P = \texttt{no}_{\texttt{collision}}$

Abstract process that captures every scene where P (for instance, $\texttt{no}_{\texttt{collision}}$) is respected. ($\rightarrow$ *agents might jump around !*)

Definition : safe process

$$\mathcal{T}\mathit{races}(\texttt{safe\_process sid } P) = e_{time}(\delta t_1); e_{scene}(\sigma_1); e_{time}(\delta t_2); e_{scene}(\sigma_2); \ldots$$
(for all σ_i st P sid σ_i is true)

Safety Properties

Example : for autonomous vehicules, $P = \texttt{no}_{\texttt{collision}}$

Abstract process that captures every scene where P (for instance, $\texttt{no}_{\texttt{collision}}$) is respected. ($\rightarrow$ *agents might jump around !*)

Definition : safe process

$$\mathcal{T}races(\texttt{safe\_process sid } P) = e_{time}(\delta t_1); e_{scene}(\sigma_1); e_{time}(\delta t_2); e_{scene}(\sigma_2); \ldots$$
(for all σ_i st P sid σ_i is true)

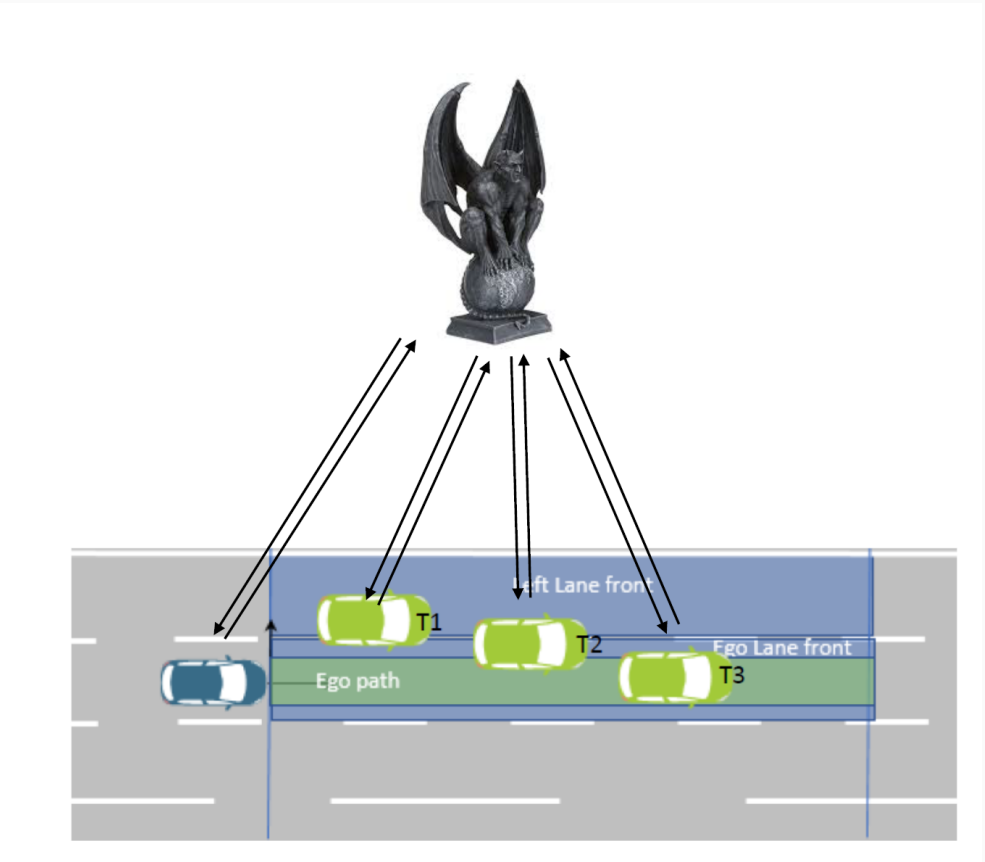
Definition : Safety of a scenario

Safety of a scenario is defined as process refinement $\leq$:

$$\texttt{safe\_process sid } P \leq \texttt{ scenario sid motion } \Delta t\ \sigma_0$$

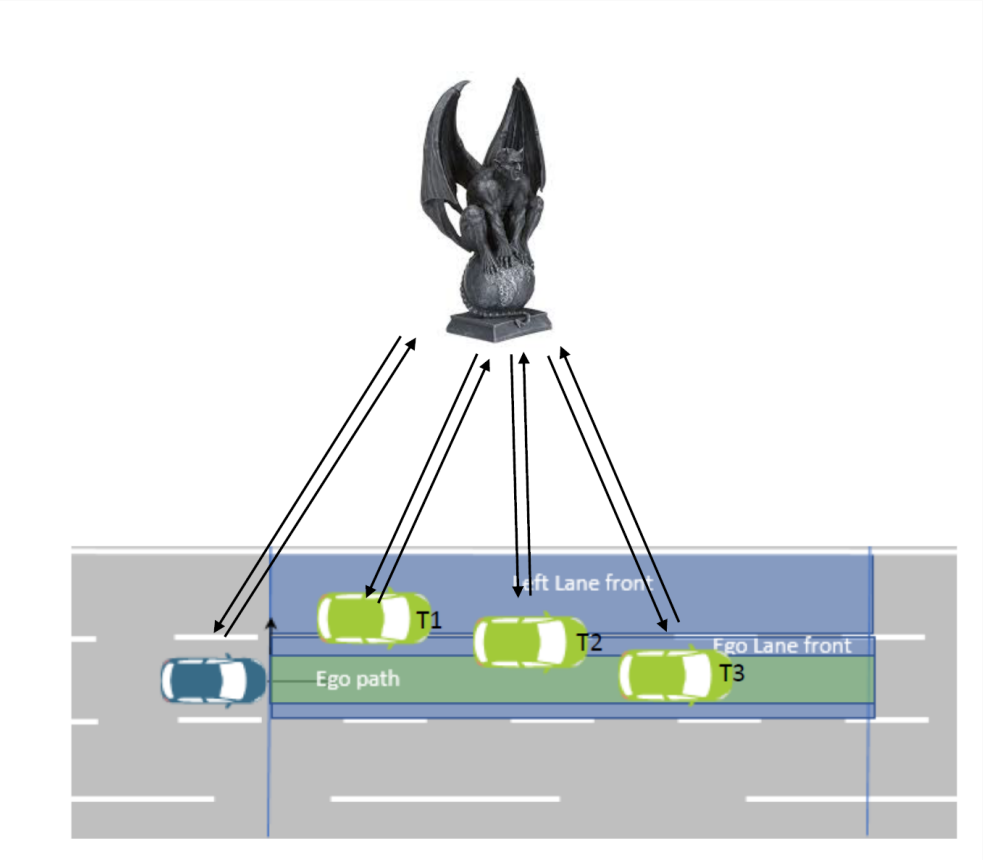
$\rightarrow \texttt{safe\_process sid no}_{\texttt{collision}}$ is the collision-free world.

Example : Autonomous Cars



- **Property :** Avoid collisions (at all costs !)
- **Invariant :** ?

Example : Autonomous Cars



- **Property :** Avoid collisions (at all costs !)
- **Invariant :** RSS-based safety distance preservation

Invariant-based Proofs

Main Theorem

If, for all σ, P_1 sid $\sigma \Rightarrow P_2$ sid σ, then :

$$\texttt{safe\_process sid } P_2 \leq \texttt{safe\_process sid } P_1$$

Property

If :

1. P sid σ_0
2. For all $\sigma, \delta t < \Delta t$, P sid σ implies P sid (motion σ δt)

Then :

$$\texttt{safe\_process sid } P \leq \texttt{scenario sid motion } \Delta t\ \sigma_0$$

Invariant-based Proofs

Main Theorem

If, for all σ, P_1 sid $\sigma \Rightarrow P_2$ sid σ, then :

$$\texttt{safe\_process sid } P_2 \leq \texttt{safe\_process sid } P_1$$

Corollary

If, for some P_{inv} :

1. For all σ, P_{inv} sid $\sigma \Rightarrow P$ sid σ
2. P_{inv} sid σ_0
3. For all $\sigma, \delta t < \Delta t$, P_{inv} sid σ implies P_{inv} sid (motion σ δt)

Then :

$$\texttt{safe\_process sid } P \leq \texttt{scenario sid motion } \Delta t\ \sigma_0$$

Framework formalized in Isabelle/HOL

$\rightarrow$ Why this approach ? We lose a lot !

- **Untractable**.
- **Hard**.

Framework formalized in Isabelle/HOL

→ Why this approach ? We lose a lot !

- **Untractable**.
- **Hard**.

But we gain a lot :

- **Exact**. *No approximations* : transcendantal functions. . . ⋆
- No **differential equation** solving ⋆.
- No potential limit to the **complexity** of scenario, number of agents, external forces, etc.
- `HOL-Analysis`
- **Oversampling** (next slide).

Framework formalized in Isabelle/HOL

→ Why this approach ? We lose a lot !

- **Untractable**.
- **Hard**.

But we gain a lot :

- **Exact**. *No approximations* : transcendantal functions. . . ⋆
- No **differential equation** solving ⋆.
- No potential limit to the **complexity** of scenario, number of agents, external forces, etc.
- `HOL-Analysis`
- **Oversampling** (next slide).

⋆ *As long as you can prove the invariant !*

Other goals

→ We get, **for free**, properties when removing agents :

$$\texttt{demon}||\texttt{agent}_1||\texttt{agent}_2 \leq \texttt{demon}||\texttt{agent}_1$$

→ Explore relationship between **driving strategies** and **kinematics** :

$$\texttt{agent}_1(\texttt{kinematics}_1, \texttt{strategy}_1) \leq \texttt{agent}_1(\texttt{kinematics}_2, \texttt{strategy}_2)?$$

- for instance, if $\texttt{strategy}_1 \subseteq \texttt{strategy}_1$ (less possible choices) :

$$\texttt{agent}_1(\texttt{kinematics}, \texttt{strategy}_1) \leq \texttt{agent}_1(\texttt{kinematics}, \texttt{strategy}_2)$$

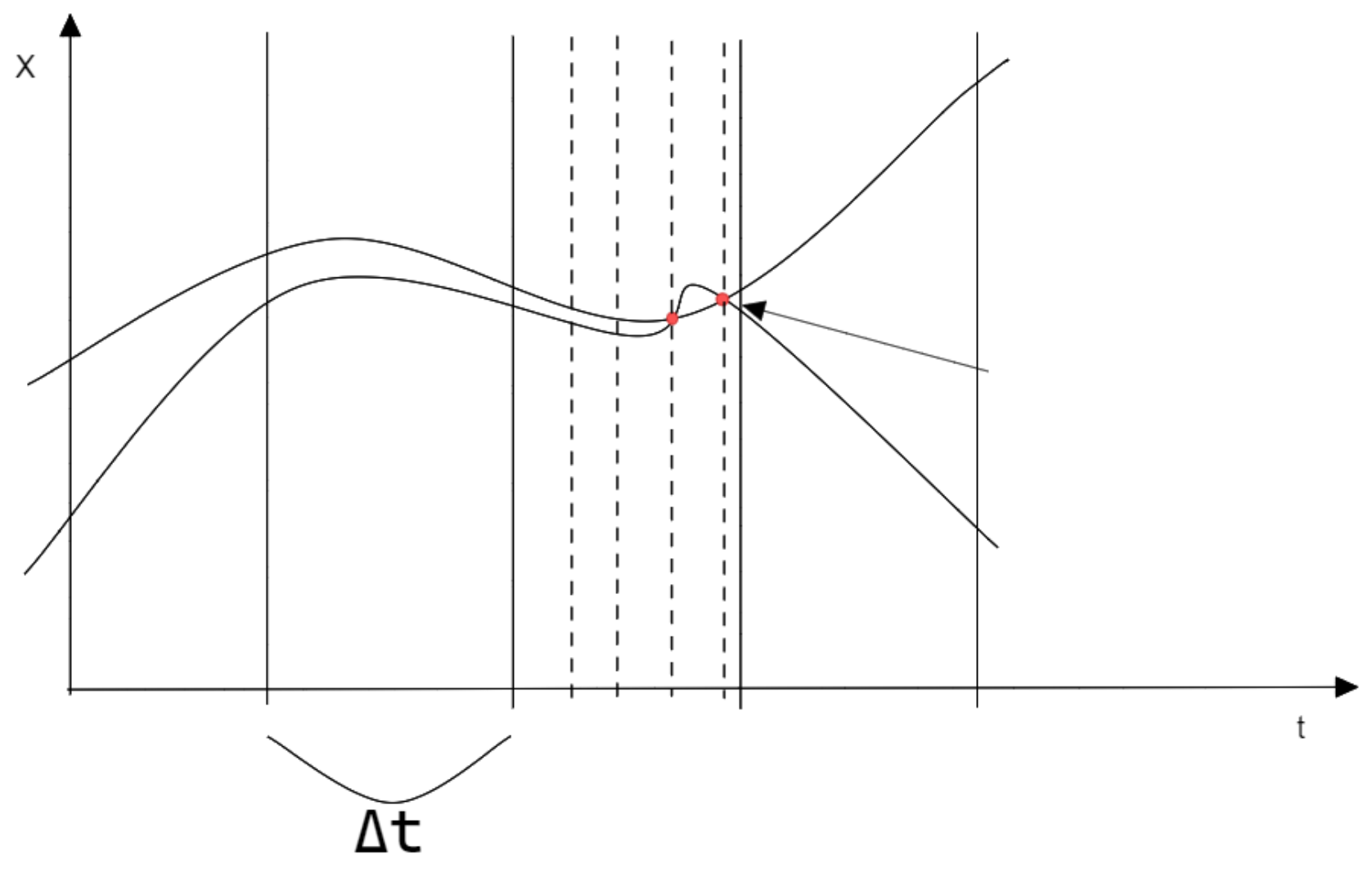
- And thus :

$$\texttt{scenario}_1 \ \cdots \leq \texttt{scenario}_2 \ \ldots$$

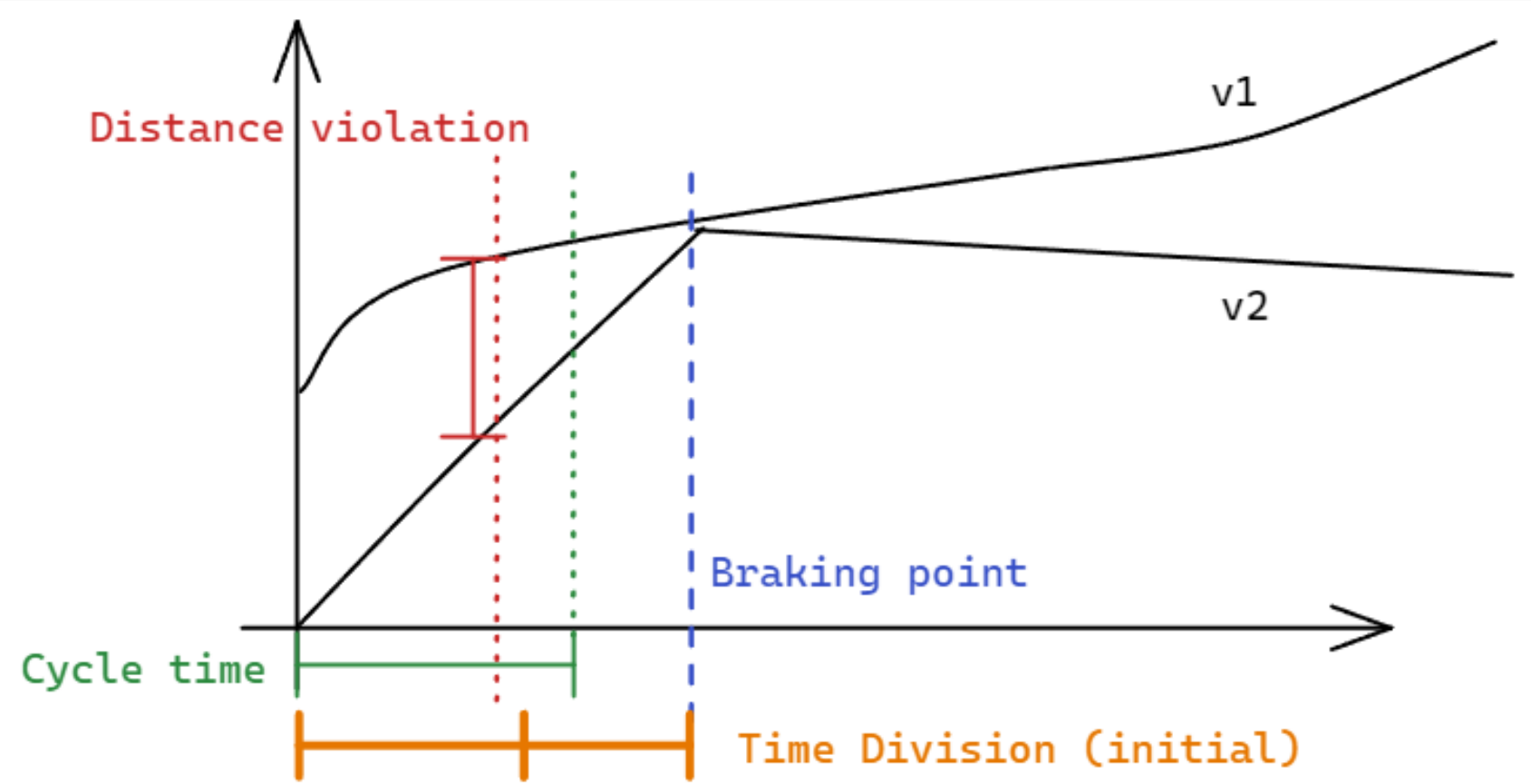
→ **Combining** kinematics, strategies and **properties** in multiple dimensions :

$$\texttt{kinematics}_x \bigoplus \texttt{kinematics}_y = \texttt{kinematics}_{2D}$$

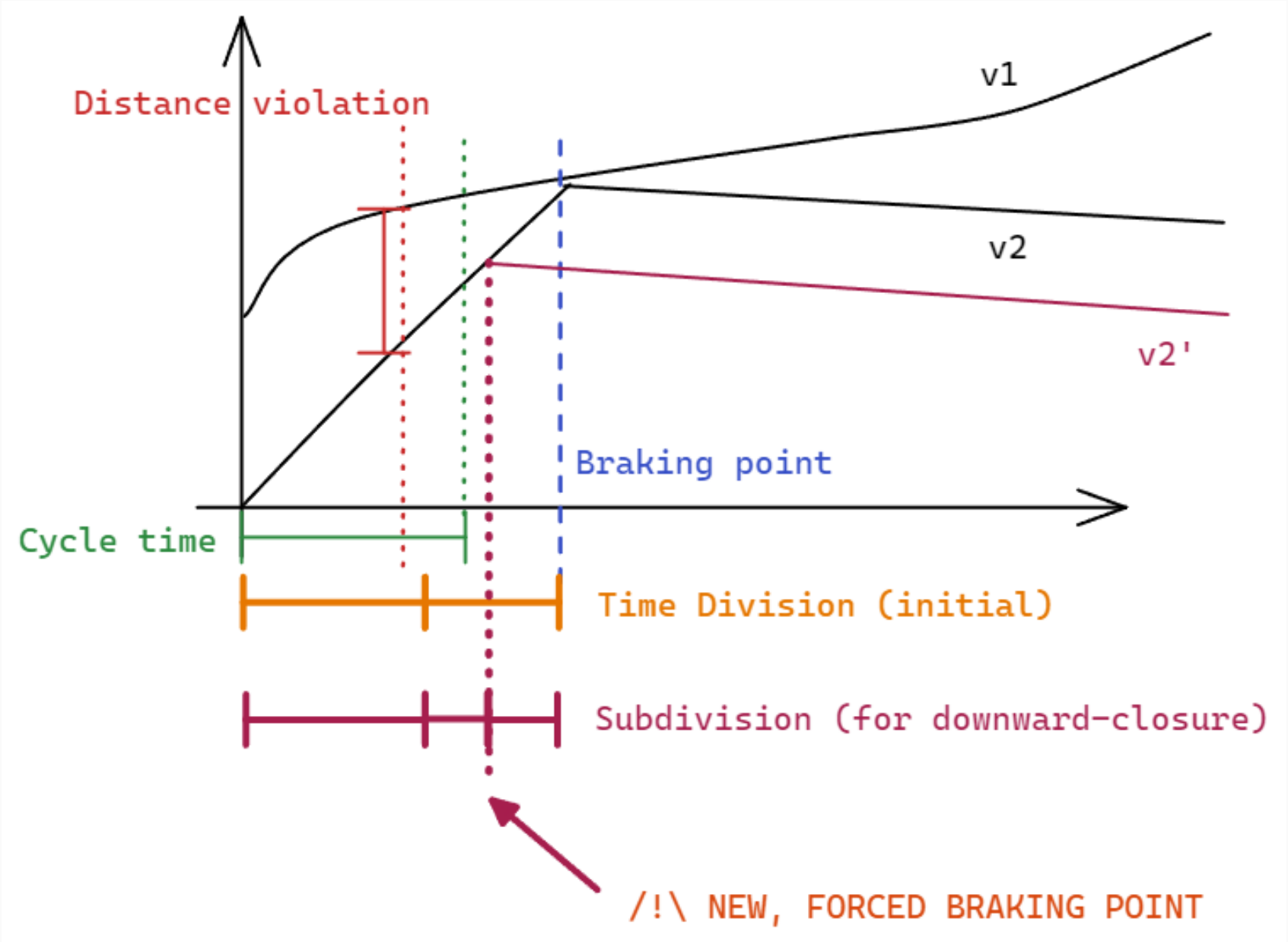
Collisions



Collisions 2



Collisions 3



Trace Stability Properties

Definition

- time $(\mathbf{T})$ is the set of all *global* times sampled on $\mathbf{T}$
- $\mathbf{T}(t)$ is the corresponding scene
- $\mathbf{T}_2$ is an oversampling of $\mathbf{T}_1$ if time$(\mathbf{T}_1) \subseteq$time$(\mathbf{T}_2)$ and, for all $t \in$ time $(\mathbf{T}_1)$, $\mathbf{T}_1(t) = \mathbf{T}_2(t)$

Conjecture

1. $\forall P$ sid. `safe_process sid` P is **closed by oversampling**

Trace Stability Properties

Definition

- time ($\mathbf{T}$) is the set of all *global* times sampled on $\mathbf{T}$
- $\mathbf{T}(t)$ is the corresponding scene
- $\mathbf{T}_2$ is an oversampling of $\mathbf{T}_1$ if time($\mathbf{T}_1$) $\subseteq$time($\mathbf{T}_2$) and, for all $t \in$ time ($\mathbf{T}_1$), $\mathbf{T}_1(t) = \mathbf{T}_2(t)$

Conjecture

1. $\forall P$ sid. `safe_process sid` P is **closed by oversampling**
2. Similarly, `scenario sid motion` Δt $\sigma_0$is **closed by oversampling**

Trace Stability Properties

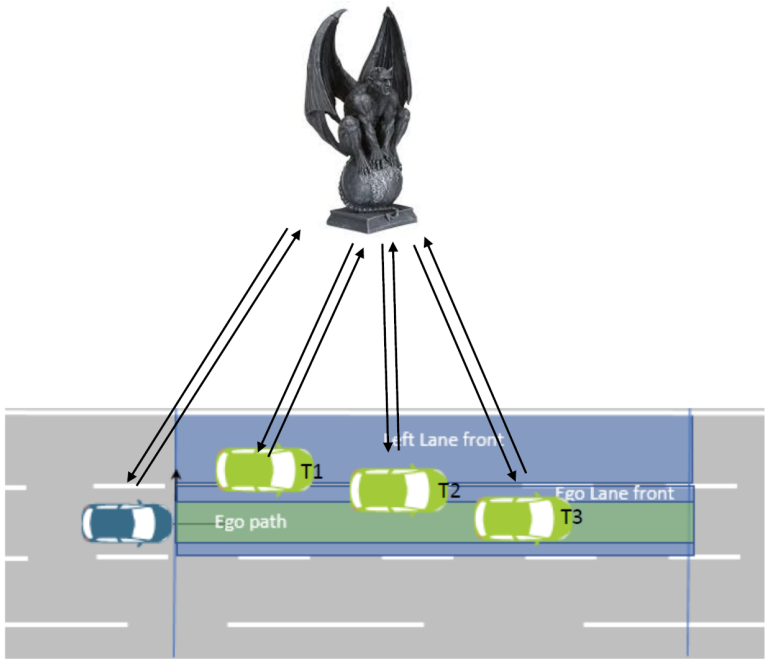
Definition

- time ($\mathbf{T}$) is the set of all *global* times sampled on $\mathbf{T}$
- $\mathbf{T}(t)$ is the corresponding scene
- $\mathbf{T}_2$ is an oversampling of $\mathbf{T}_1$ if time($\mathbf{T}_1$) $\subseteq$time($\mathbf{T}_2$) and, for all $t \in$ time ($\mathbf{T}_1$), $\mathbf{T}_1(t) = \mathbf{T}_2(t)$

Conjecture

1. $\forall P$ sid. `safe_process sid` P is **closed by oversampling**
2. ~~Similarly, `scenario sid motion` $\Delta t\ \sigma_0$ is **closed by oversampling**~~

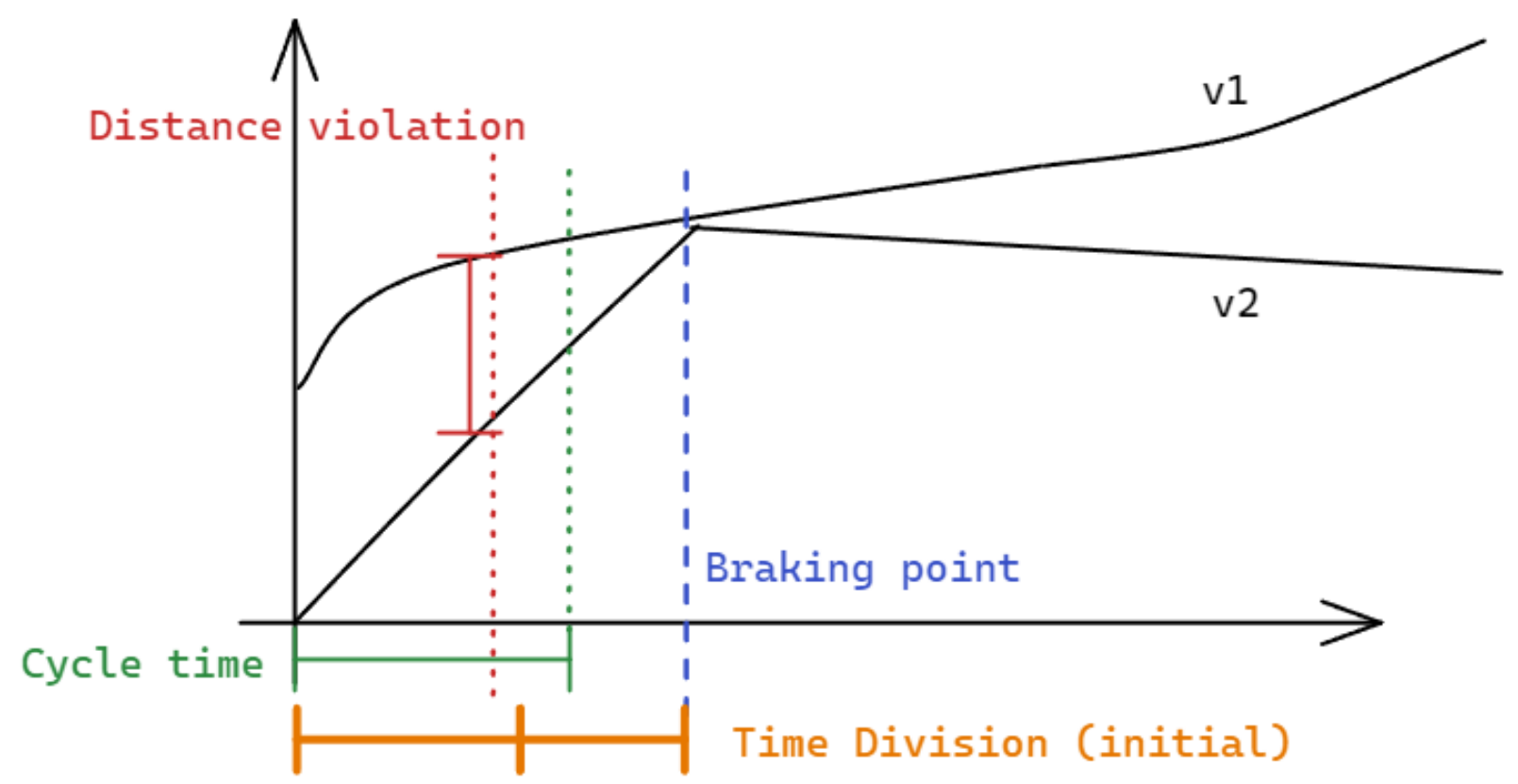
Desynchronization



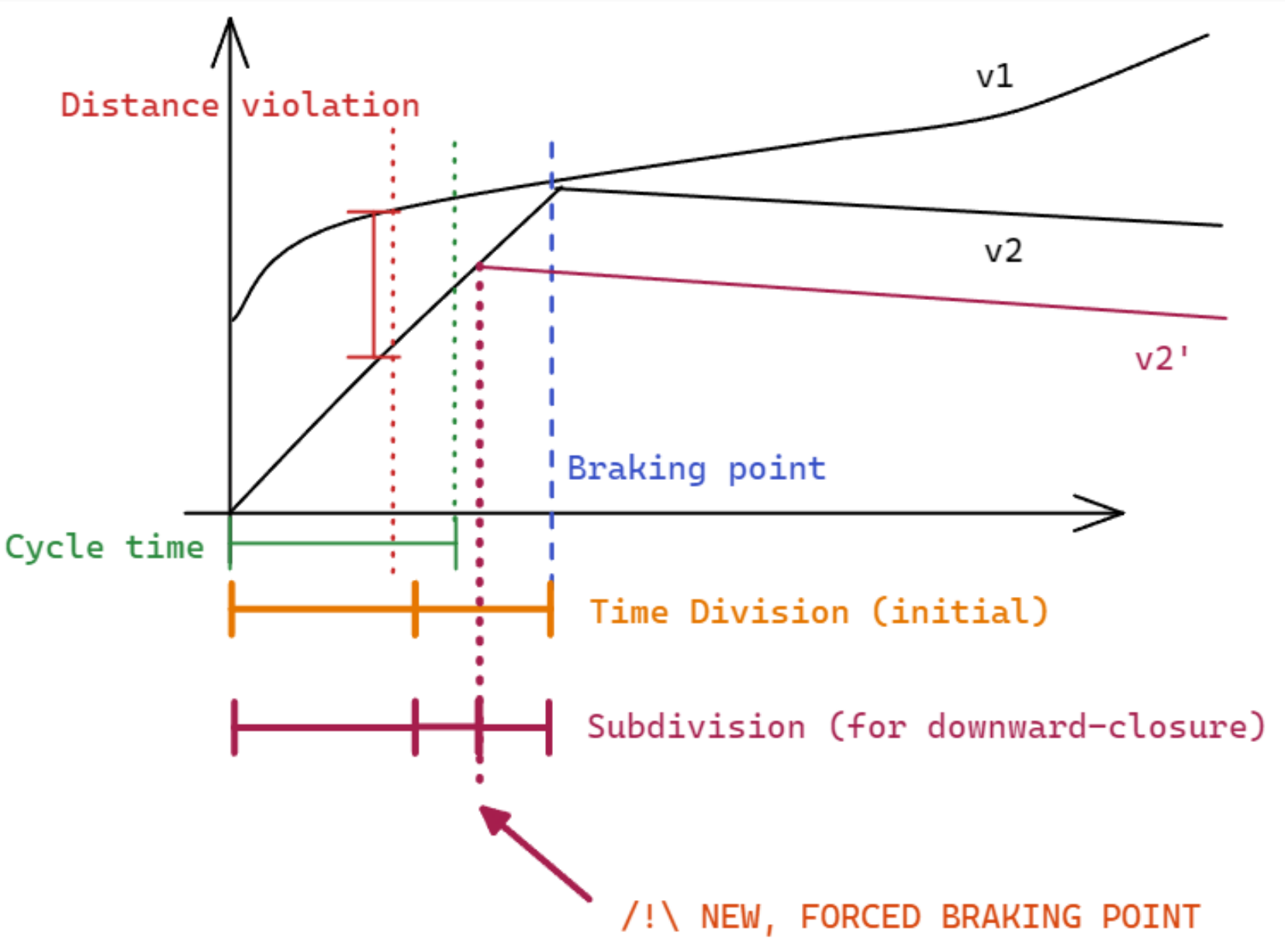
→ T_0, T_1, T_2... **shouldn't make their decisions at the same time**

→ **vehicules should be allowed to make no decision at all...** *(until their reaction time is reached)*

Cycle times


Distance violation
v1
v2
Braking point
Cycle time
Time Division (initial)

Cycle times


Distance violation
v1
v2
v2'
Braking point
Cycle time
Time Division (initial)
Subdivision (for downward-closure)
/!\ NEW, FORCED BRAKING POINT

Trace Stability Properties

Prefix sample-closeness

For any $\delta t < \Delta t$ and for any **finite** prefix T_0 of $\mathbf{T} \in \mathcal{T}races(\texttt{scenario})$, there is a **finite prefix** T_0' of $\mathbf{T}' \in \mathcal{T}races(\texttt{scenario})$ such that :

- $\text{time}\,(T_0') = \text{time}\,(T_0) + \delta t$
- T_0' is an extension of T_0

Trace Stability Properties

Prefix sample-closeness

For any $\delta t < \Delta t$ and for any **finite** prefix T_0 of $\mathbf{T} \in \mathcal{T}races(\texttt{scenario})$, there is a **finite prefix** T_0' of $\mathbf{T}' \in \mathcal{T}races(\texttt{scenario})$ such that :

- time $(T_0') =$ time $(T_0) + \delta t$
- T_0' is an extension of T_0

With prefix closeness, we can always 'zoom-in' to check if there is no collision !