

Aurèle Barrière

Doctorat, IRISA 

avec Sandrine Blazy et David Pichardie

*Vérification formelle de
compilation à la volée (JIT)*

2019-2022

 EAPLS Best PhD Award.

PostDoc, EPFL 

avec Clément Pit-Claudel

*Vers des moteurs de regex modernes
linéaires et vérifiés*

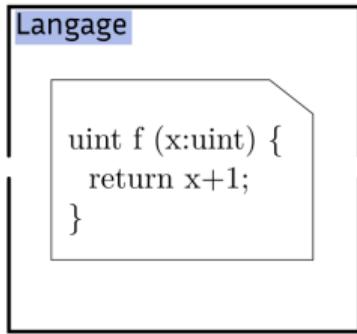
2023-2025

Stages : SNU  (2017), Federico II  (2017), Princeton  (2018).

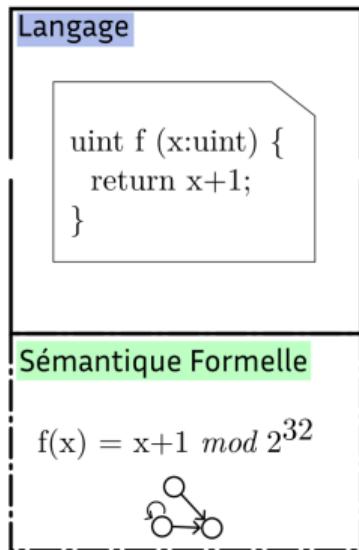
Intégration : Équipe SyCoMoRES

Comment faire confiance à l'exécution d'un programme ?

Comment faire confiance à l'exécution d'un programme ?

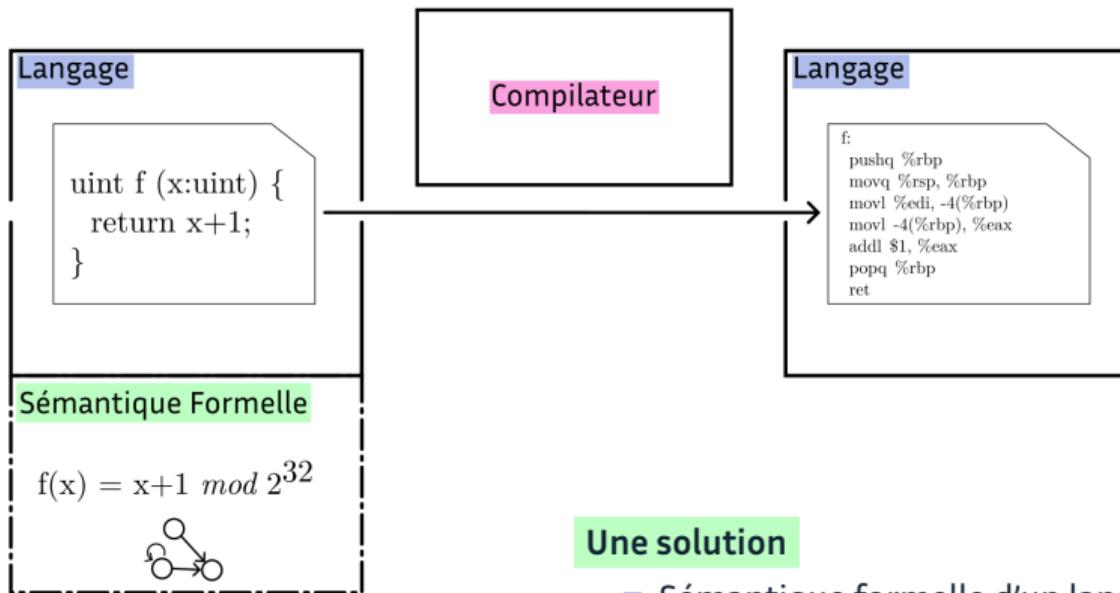


Comment faire confiance à l'exécution d'un programme ?

**Une solution**

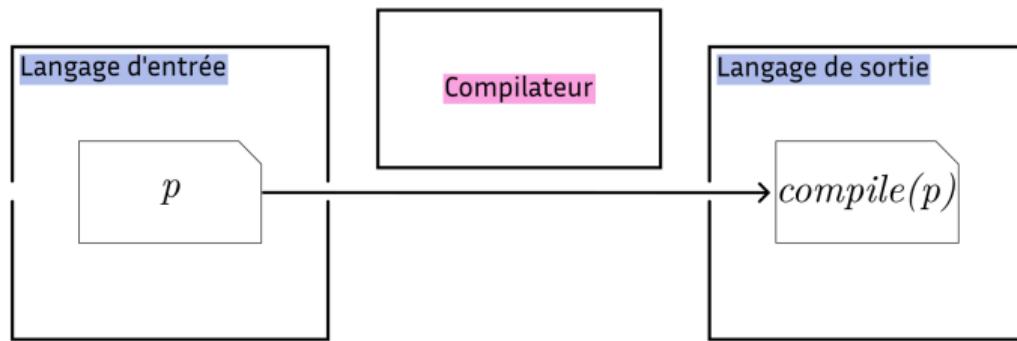
- Sémantique formelle d'un langage

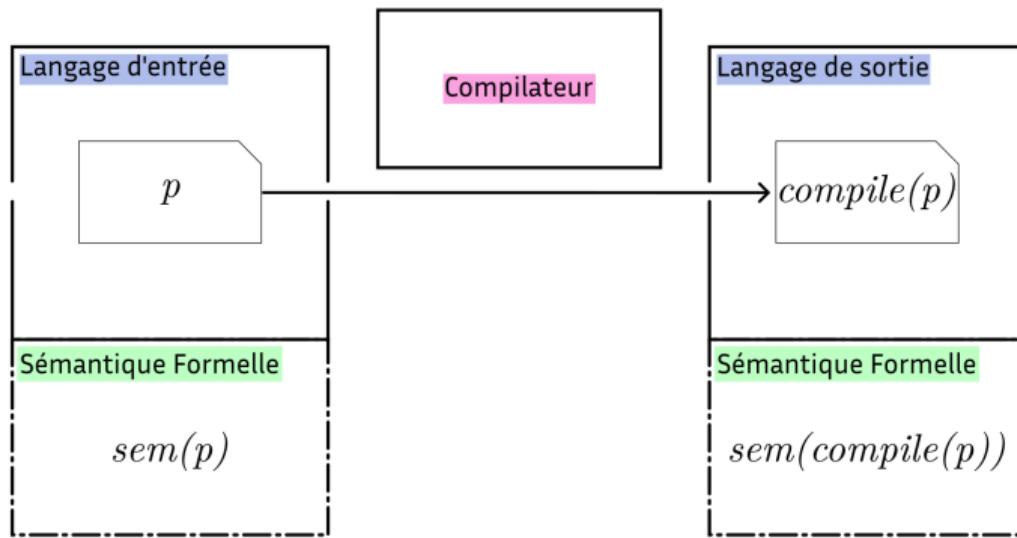
Comment faire confiance à l'exécution d'un programme ?

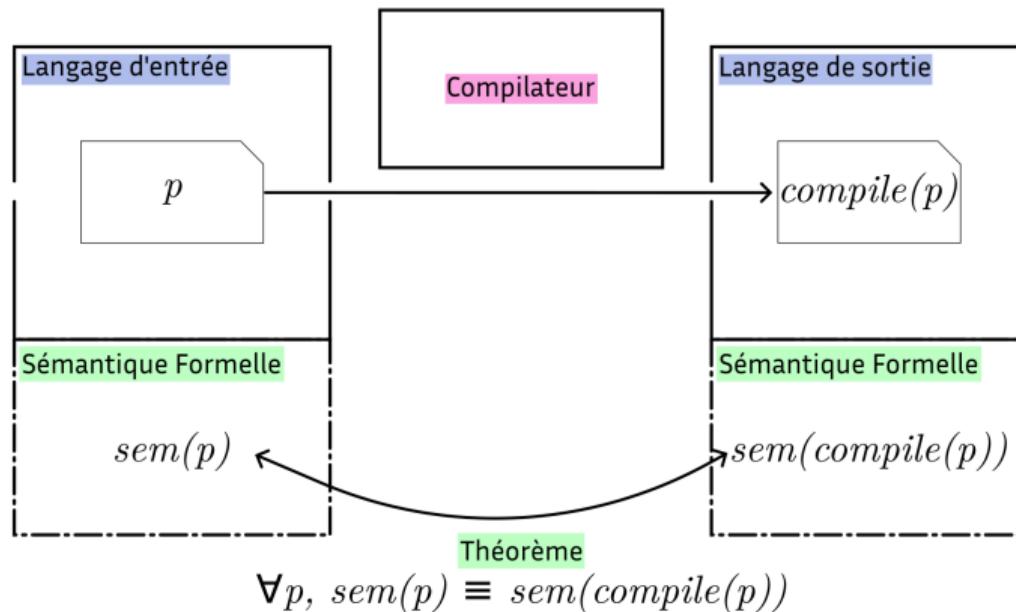


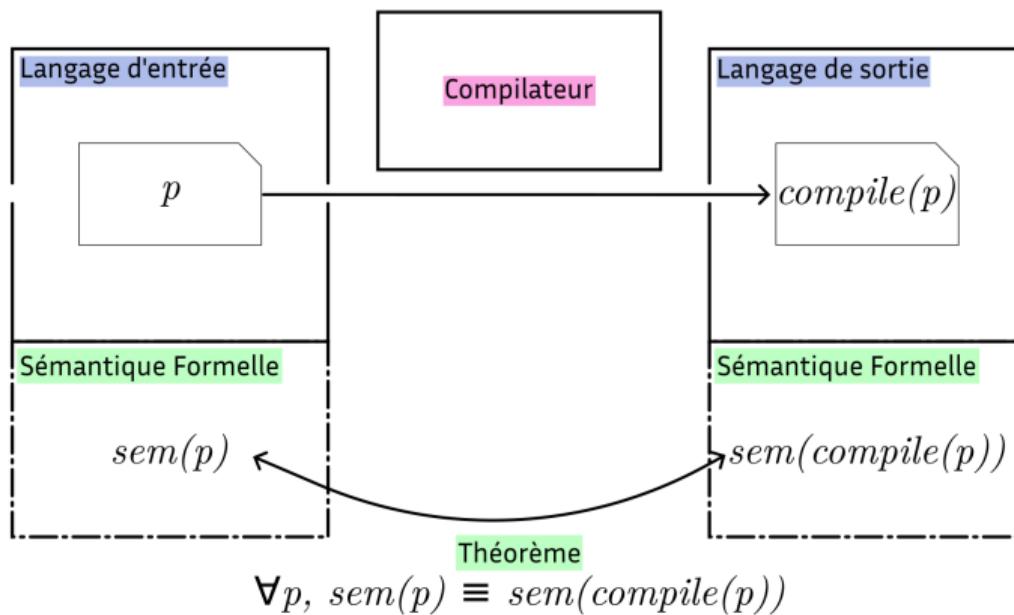
Une solution

- Sémantique formelle d'un langage
- Théorie formelle de la compilation









Compilateurs formellement vérifiés dans un assistant de preuve

CompCert (Coq/Rocq) [Leroy, POPL'2006], CakeML (HOL) [Kumar et al. POPL'2014]...
[Yang et al. PLDI'2011] : Des centaines de bugs dans GCC et LLVM, aucun dans CompCert.

Comment faire confiance à l'exécution d'un programme sur le web?

Comment faire confiance à l'exécution d'un programme sur le web?

Un besoin de garanties

- Les navigateurs sont des environnements d'exécution, pour JavaScript et WebAssembly.
- Leurs bugs sont dangereux! Google Chrome et Firefox en 2025 : [\[CVE-2025-0291\]](#),
[\[CVE-2025-0434\]](#), [\[CVE-2025-0445\]](#), [\[CVE-2025-0611\]](#), [\[CVE-2025-0612\]](#), [\[CVE-2025-0995\]](#),
[\[CVE-2025-0998\]](#), [\[CVE-2025-0999\]](#), [\[CVE-2025-1011\]](#), [\[CVE-2025-1914\]](#), [\[CVE-2025-1933\]](#).

Comment faire confiance à l'exécution d'un programme sur le web?

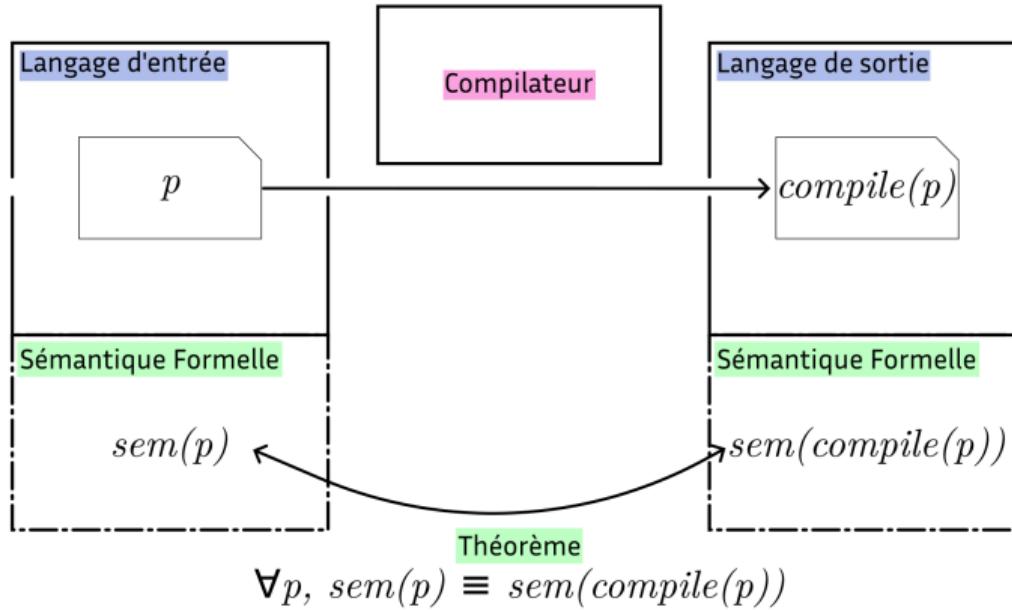
Un besoin de garanties

- Les navigateurs sont des environnements d'exécution, pour JavaScript et WebAssembly.
- Leurs bugs sont dangereux! Google Chrome et Firefox en 2025 : [\[CVE-2025-0291\]](#),
[\[CVE-2025-0434\]](#), [\[CVE-2025-0445\]](#), [\[CVE-2025-0611\]](#), [\[CVE-2025-0612\]](#), [\[CVE-2025-0995\]](#),
[\[CVE-2025-0998\]](#), [\[CVE-2025-0999\]](#), [\[CVE-2025-1011\]](#), [\[CVE-2025-1914\]](#), [\[CVE-2025-1933\]](#).

Un problème

Les techniques de compilation et d'exécution utilisées ont largement dévié de la théorie.

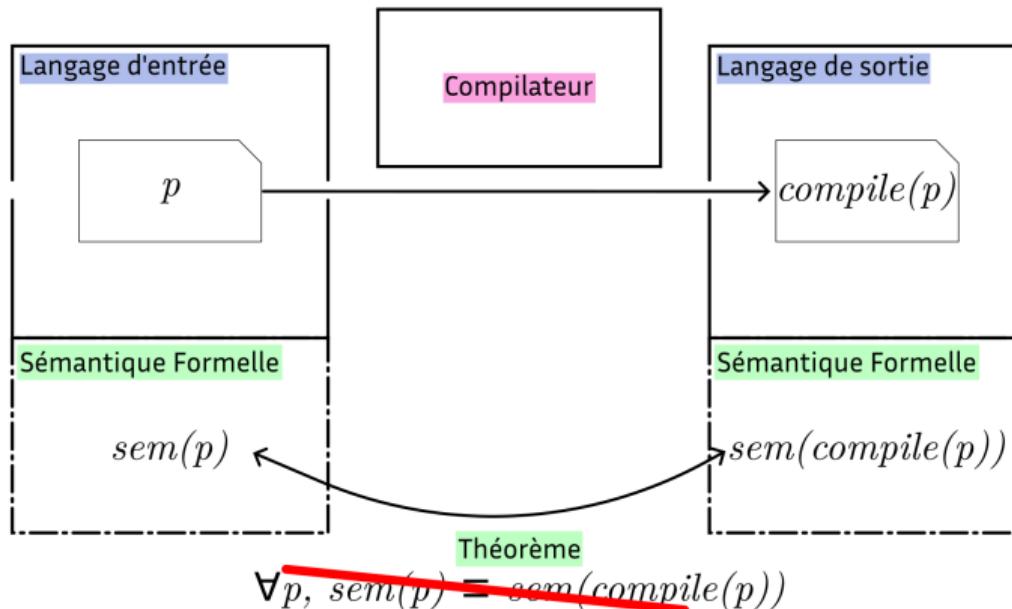
Comment faire confiance à l'exécution d'un programme sur le web?



Un problème

Les techniques de compilation et d'exécution utilisées ont largement dévié de la théorie.

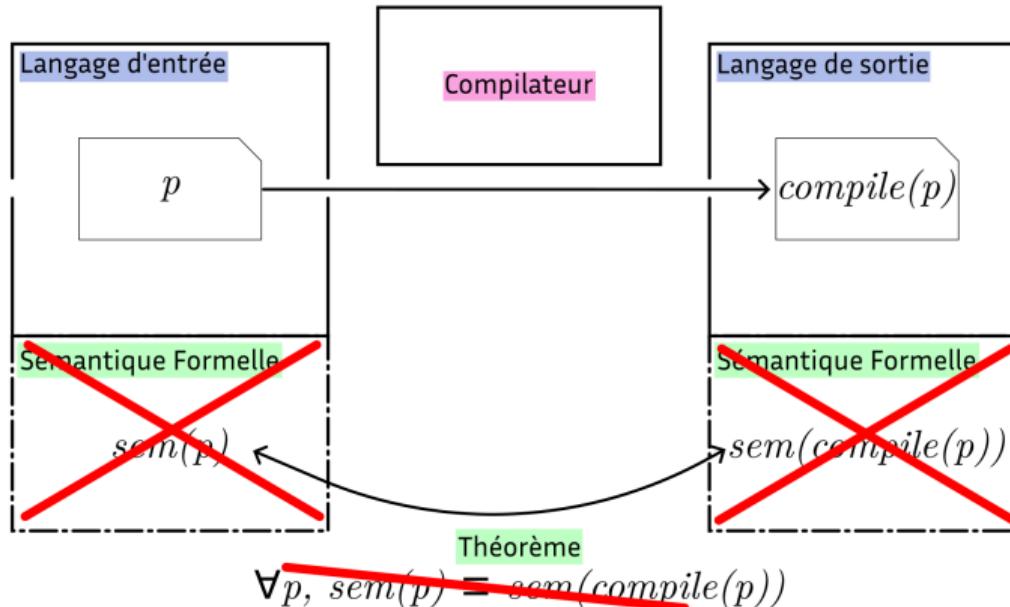
Comment faire confiance à l'exécution d'un programme sur le web?



Un problème

Les techniques de compilation et d'exécution utilisées ont largement dévié de la théorie.

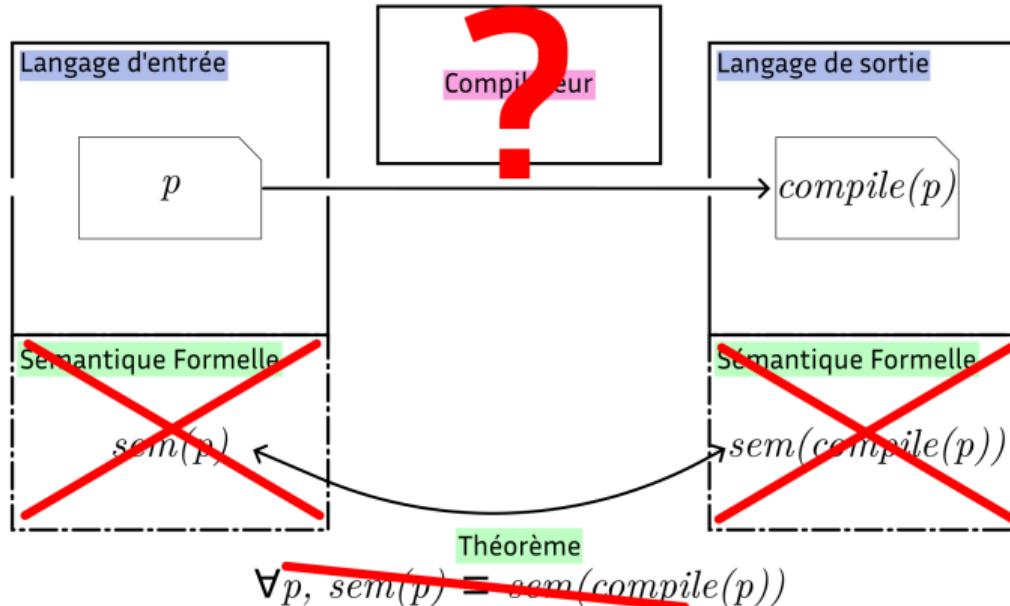
Comment faire confiance à l'exécution d'un programme sur le web?



Un problème

Les techniques de compilation et d'exécution utilisées ont largement dévié de la théorie.

Comment faire confiance à l'exécution d'un programme sur le web?



Un problème

Les techniques de compilation et d'exécution utilisées ont largement dévié de la théorie.

Concevoir à la fois la théorie et des implémentations vérifiées pour un web de confiance.

Concevoir à la fois la théorie et des implémentations vérifiées pour un web de confiance.

Deux cas de compilation non traditionnelle du web

Doctorat : Vérification formelle de compilation à la volée (JIT)

PostDoc : Étude formelle des regex JavaScript

Concevoir à la fois la théorie et des implémentations vérifiées pour un web de confiance.

Deux cas de compilation non traditionnelle du web

Doctorat : Vérification formelle de compilation à la volée (JIT)

PostDoc : Étude formelle des regex JavaScript

Bénéfices	JIT	Regex
Implémentations de confiance	Prototypes de JIT	PikeVM (en cours)
Comprendre les techniques modernes	Optimisations dynamiques	Futurs uniformes
Concevoir de nouvelles techniques	Instructions spéculatives	Nouveaux algorithmes linéaires

Concevoir à la fois la théorie et des implémentations vérifiées pour un web de confiance.

Deux cas de compilation non traditionnelle du web

Doctorat : Vérification formelle de compilation à la volée (JIT)

PostDoc : Étude formelle des regex JavaScript

Bénéfices	JIT	Regex
Implémentations de confiance	Prototypes de JIT	PikeVM (en cours)
Comprendre les techniques modernes	Optimisations dynamiques	Futurs uniformes
Concevoir de nouvelles techniques	Instructions spéculatives	Nouveaux algorithmes linéaires

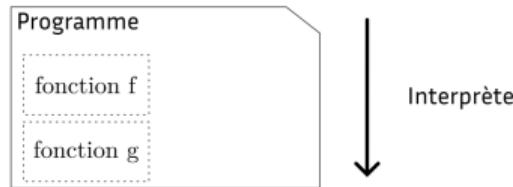
JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.

Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.

JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.

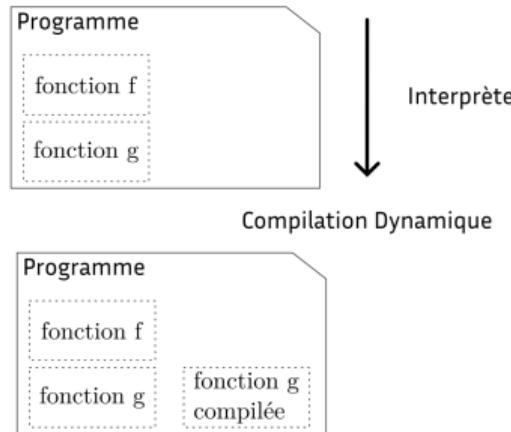


Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.

JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.

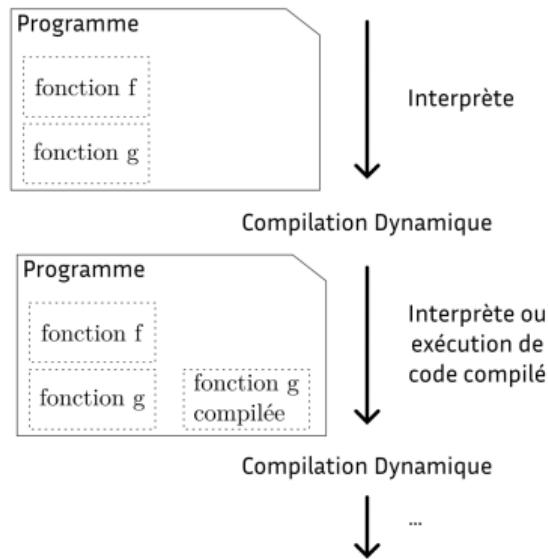


Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.

JIT (Just-in-Time) = entremêler **exécution** et **compilation** du programme.



Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.

JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.



Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.

JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.



Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.

JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.

CVE-2019-11707, CVE-2019-11708: Multiple Zero-Day Vulnerabilities in Mozilla Firefox Exploited in the Wild

Satnam Narang | June 18, 2019 | 3 Min Read | Twitter | Facebook | LinkedIn

Security researchers discover two zero-day vulnerabilities in Mozilla Firefox used in targeted attacks.

Les dangers du JIT

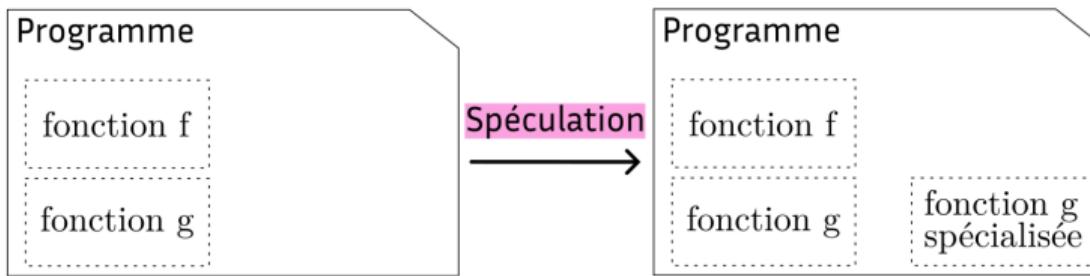
Un JIT génère du code exécutable ! 2019 : Coinbase est victime d'une attaque.

Comment écrire un compilateur JIT correct ?

Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

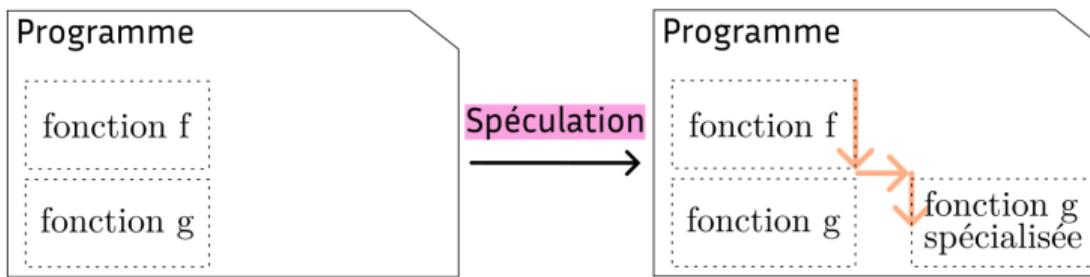
Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

Collaboration : Olivier Flückiger & Jan Vitek (Northeastern 🇺🇸), créateurs du JIT Rir pour le langage R.



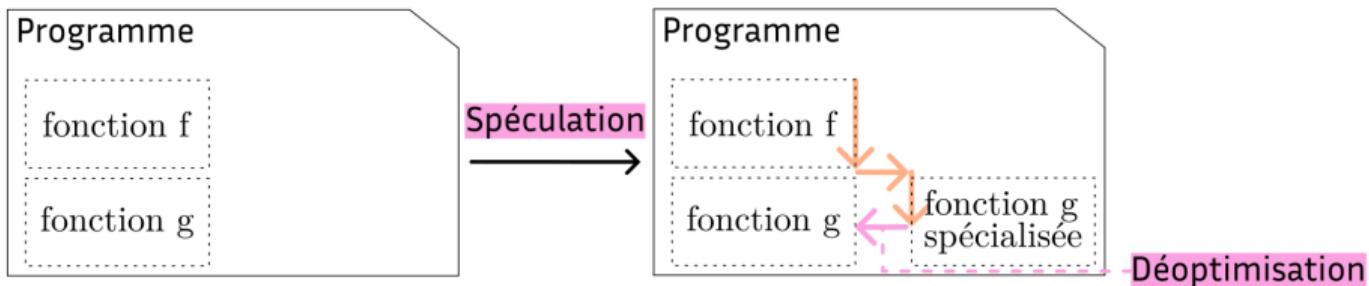
Spéculation

Compiler des versions spécialisées de fonctions.



Spéculation

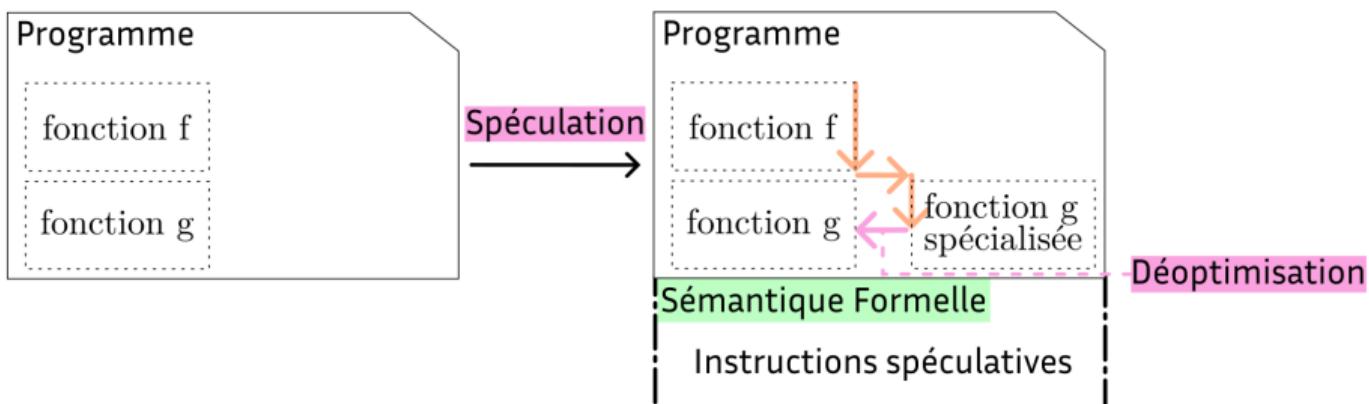
Compiler des versions spécialisées de fonctions.



Spéculation

Compiler des versions spécialisées de fonctions.

Déoptimisation : sauter dynamiquement de la fonction spécialisée et compilée vers la version originale.



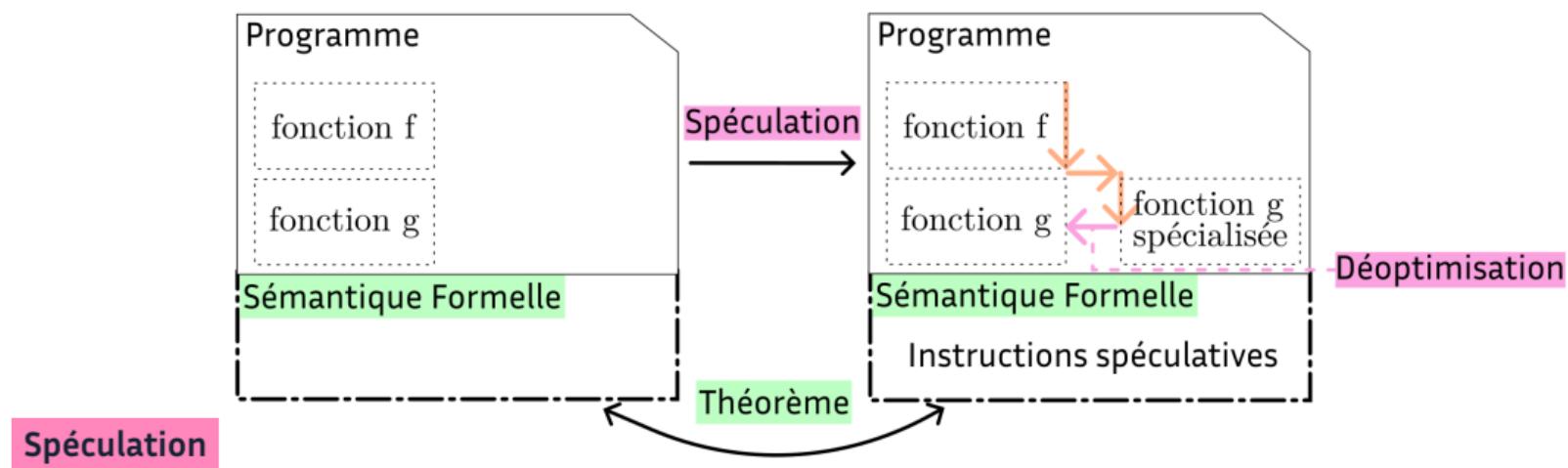
Spéculation

Compiler des versions spécialisées de fonctions.

Déoptimisation : sauter dynamiquement de la fonction spécialisée et compilée vers la version originale.

Contributions

Sémantique formelle pour des instructions spéculatives (difficulté : non déterminisme).



Compiler des versions spécialisées de fonctions.

Déoptimisation : sauter dynamiquement de la fonction spécialisée et compilée vers la version originale.

Contributions

Sémantique formelle pour des instructions spéculatives (difficulté : non déterminisme).

Vérification de leur insertion, manipulation et compilation.

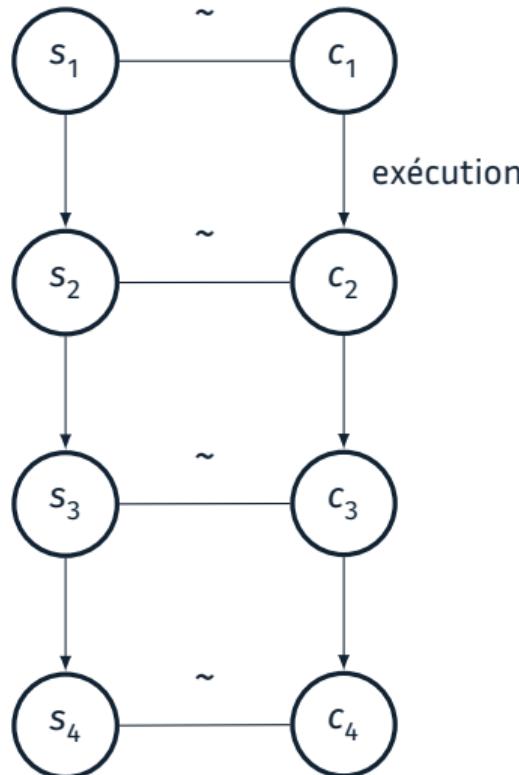
Une méthode de référence pour spéculer dans un JIT.

Programme SourceProgramme Compilé**Simulation :** (simplifiée)

- concevoir un invariant ~ (relation entre états sémantiques).
- montrer que l'invariant est préservé à chaque pas.

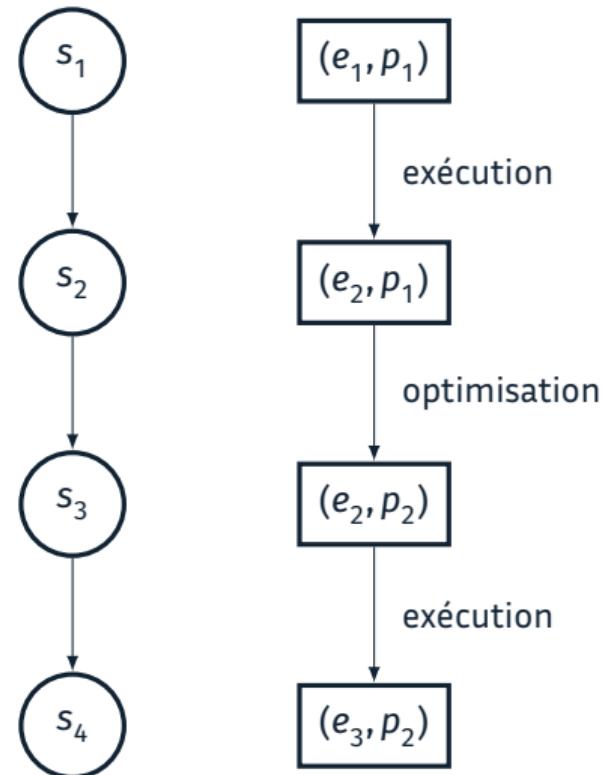
Programme SourceProgramme Compilé**Simulation :** (simplifiée)

- concevoir un invariant ~ (relation entre états sémantiques).
- montrer que l'invariant est préservé à chaque pas.

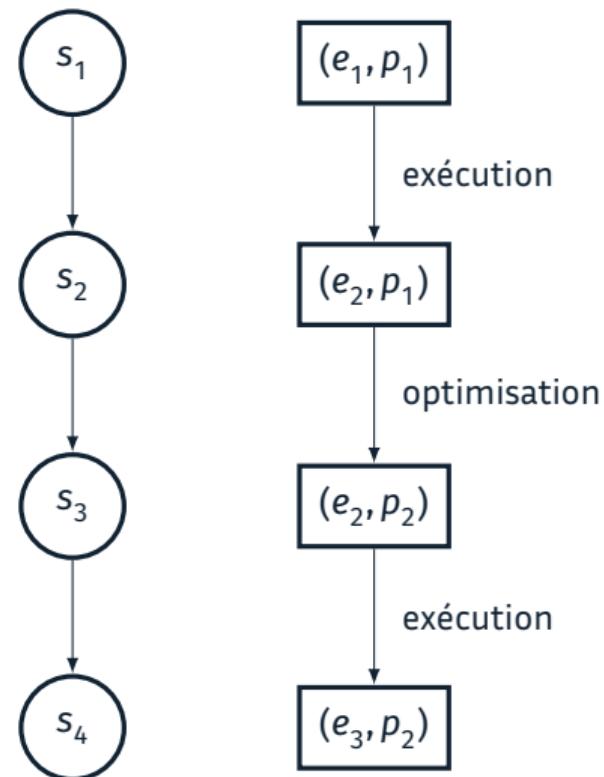


Programme SourceExécution JIT**Simulation :** (simplifiée)

- concevoir un invariant ~ (relation entre états sémantiques).
- montrer que l'invariant est préservé à chaque pas.



Problème : dans un JIT, le programme change.

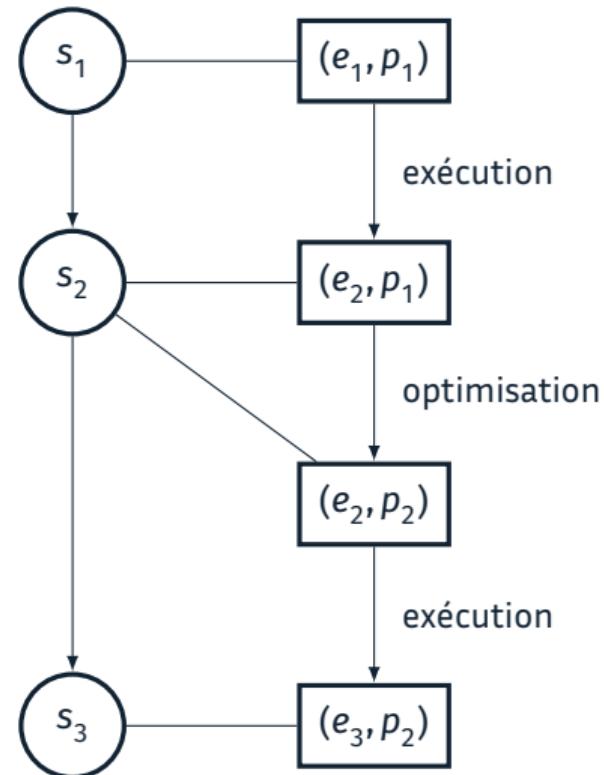
Programme SourceExécution JIT**Simulation :** (simplifiée)

- concevoir un invariant ~ (relation entre états sémantiques).
- montrer que l'invariant est préservé à chaque pas.

Problème : dans un JIT, le programme change.

Idée clé : à tout moment, le programme du JIT doit être équivalent au programme original.

Cette équivalence peut s'exprimer avec une simulation !

Programme SourceExécution JIT**Simulation :** (simplifiée)

- concevoir un invariant ~ (relation entre états sémantiques).
- montrer que l'invariant est préservé à chaque pas.

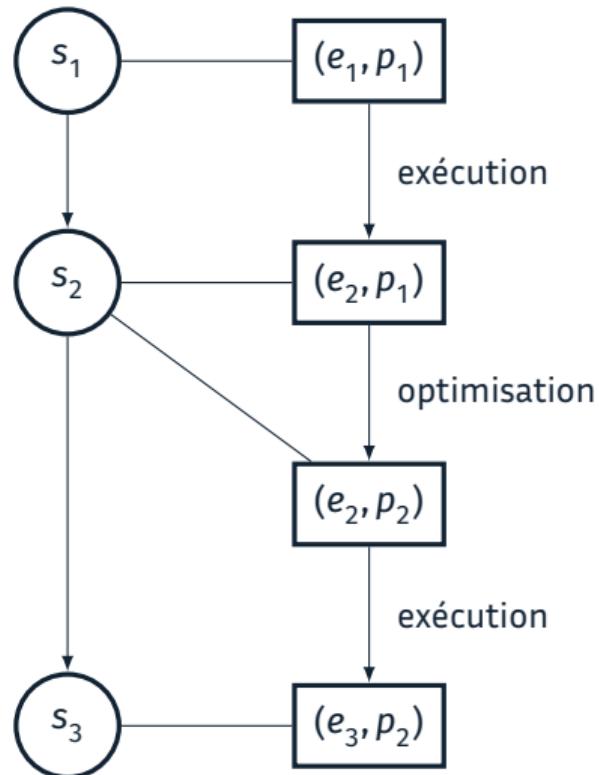
Problème : dans un JIT, le programme change.

Idée clé : à tout moment, le programme du JIT doit être équivalent au programme original.

Cette équivalence peut s'exprimer avec une simulation !

Solution : Une simulation dont l'invariant est une simulation.

$s \sim (e, p)$ ssi $\exists \sim_i$, p et p_1 sont simulés avec $\sim_i \wedge s \sim_i e$.

Programme SourceExécution JIT**Simulations Imbriquées, non simplifiées :****(1) Initialisation dynamique**
 $\forall s_y, \text{ si } s_y \text{ est un état de synchronisation, alors } s_y \sim_{int} s_y$
(2) Préservation de progrès

$$\forall s_1 s'_1 t s_2, s_1 \sim_{int} s_2 \wedge s_1 \xrightarrow[p_1]{t} s'_1 \implies \exists t' s'_2, s_2 \xrightarrow[p]{t'} s'_2$$

(3) Diagramme interne

$$\forall s_1 s_2 s'_2 t, s_1 \sim_{int} s_2 \wedge s_2 \xrightarrow[p]{t} s'_2 \implies (\exists s'_1, s_1 \xrightarrow[p_1]{t} s'_1 \wedge s'_1 \sim_{int} s'_2) \vee$$

$$(s_1 \sim_{int} s'_2 \wedge m_{int}(s'_2) < m_{int}(s_2) \wedge t = \emptyset)$$

Simulation Interne $\sim_{int} m_{int} p_1 p$

$s \sim_{int} e$

Simulation Interne $\sim_{int} m_{int} p_1 p$

$$s \sim_{ext} (e, p, n, ps)$$

Mon doctorat

Des prototypes de JITs vérifiés et exécutables.

Artéfacts :

+30K lignes de Coq/Rocq

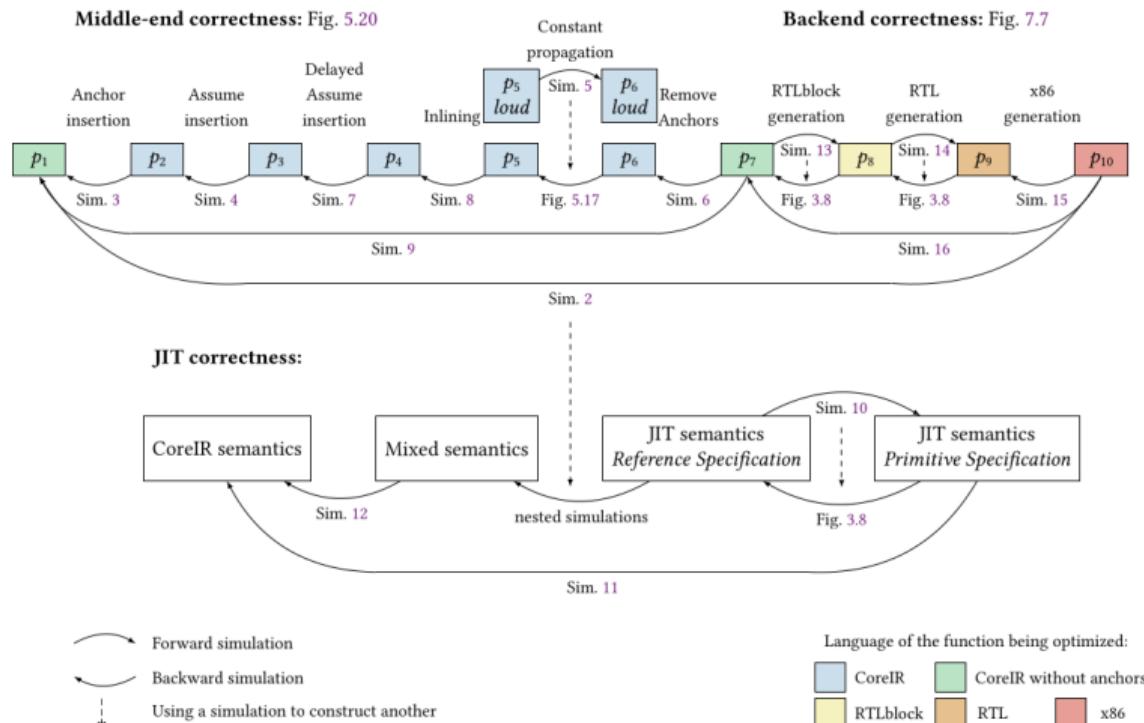


Figure 8.1 – Composing all our simulations for an effectful JIT with speculation and native code generation

$r ::=$	a	Caractère
	$r_1 r_2$	Séquence
	$r_1 r_2$	Disjonction
	r^*	Étoile

$r ::= a$	Caractère
$r_1 r_2$	Séquence
$r_1 r_2$	Disjonction
r^*	Étoile
<hr/>	
[a - z]	Classe de caractères
\$, ^	Ancre
(r)	Groupe de capture
(?= r)	Lookahead
(?<= r)	Lookbehind
\1	Backreference

Complexité : linéaire , inconnue , NP-dur .

Exemple de Lookahead : a(?=b) matche les "a", seulement s'ils sont suivis d'un "b".

$r ::= a$	Caractère
$r_1 r_2$	Séquence
$r_1 r_2$	Disjonction
r^*	Étoile
<hr/>	<hr/>
$[a - z]$	Classe de caractères
$\$, ^$	Ancre
(r)	Groupe de capture
$(?=r)$	Lookahead
$(?<=r)$	Lookbehind
$\backslash 1$	Backreference

Problème : complexité exponentielle

Vulnérabilité ReDoS : 12% des serveurs JS vulnérables.
`"a".repeat(100).match(/(a*)*b/)`: 10^{14} ans.

Complexité : linéaire , inconnue , NP-dur .

Exemple de Lookahead : `a(?=b)` matche les "a", seulement s'ils sont suivis d'un "b".

$r ::= a$	Caractère
$r_1 r_2$	Séquence
$r_1 r_2$	Disjonction
r^*	Étoile
<hr/>	
$[a - z]$	Classe de caractères
$$, ^$	Ancre
(r)	Groupe de capture
$(?=r)$	Lookahead
$(?<=r)$	Lookbehind
$\backslash 1$	Backreference

Problème : complexité exponentielle

Vulnérabilité ReDoS : 12% des serveurs JS vulnérables.
`"a".repeat(100).match(/(a*)*b/)`: 10^{14} ans.

 Solution dans V8 (Google Chrome/Node.JS) :
un moteur linéaire pour les fonctionnalités linéaires.

Complexité : linéaire, inconnue, NP-dur.

Exemple de Lookahead : `a(?=b)` matche les "a", seulement s'ils sont suivis d'un "b".

$r ::= a$	Caractère
$r_1 r_2$	Séquence
$r_1 r_2$	Disjonction
r^*	Étoile
<hr/>	
$[a - z]$	Classe de caractères
$$, ^$	Ancre
(r)	Groupe de capture
$(?= r)$	Lookahead
$(?<= r)$	Lookbehind
$\backslash 1$	Backreference

Problème : complexité exponentielle

Vulnérabilité ReDoS : 12% des serveurs JS vulnérables.
`"a".repeat(100).match(/(a*)*b/)`: 10^{14} ans.

 Solution dans V8 (Google Chrome/Node.JS) : un moteur linéaire pour les fonctionnalités linéaires.

Des problèmes algorithmiques et sémantiques

J'ai montré que :
 Les algorithmes linéaires étaient faux ou non linéaires.
 Les modèles sémantiques étaient incomplets ou faux.

Complexité : linéaire, inconnue, NP-dur.

Exemple de Lookahead : `a(?=b)` matche les "a", seulement si ils sont suivis d'un "b".

Les spécificités sémantiques de JavaScript

- Les groupes de captures ont une sémantique unique (réinitialisation à chaque itération).
- **Nouveau :** L'étoile a une sémantique différente! “`ab`”.`match(/(a?b??)*/)`

Les spécificités sémantiques de JavaScript

- Les groupes de captures ont une sémantique unique (réinitialisation à chaque itération).
- **Nouveau :** L'étoile a une sémantique différente! “`ab`”.`match(/(a?b??)*/)`

Fonctionnalité	État de l'art linéaire (V8)	Mes nouveaux algorithmes
Quantificateurs nullables (*,+)	incorrect	$O(r \times s)$

$|r|$: taille de la regex
 $|s|$: taille de la chaîne de caractères

Les spécificités sémantiques de JavaScript

- Les groupes de captures ont une sémantique unique (réinitialisation à chaque itération).
- **Nouveau :** L'étoile a une sémantique différente ! “`ab`”.`match(/(a?b??)*/)`

Fonctionnalité	État de l'art linéaire (V8)	Mes nouveaux algorithmes
Quantificateurs nullables (*,+)	incorrect	$O(r \times s)$
Groupes de capture quantifiés	$O(r ^2 \times s)$	$O(r \times s)$
Plus non nullable	$O(2^{ r } \times s)$	$O(r \times s)$
Plus nullable greedy	$O(2^{ r } \times s)$	$O(r \times s)$

$|r|$: taille de la regex

$|s|$: taille de la chaîne de caractères

Les spécificités sémantiques de JavaScript

- Les groupes de captures ont une sémantique unique (réinitialisation à chaque itération).
- **Nouveau :** L'étoile a une sémantique différente ! “`ab`”.`match(/(a?b??)*/)`

Fonctionnalité	État de l'art linéaire (V8)	Mes nouveaux algorithmes
Quantificateurs nullables (*,+)	incorrect	$O(r \times s)$
Groupes de capture quantifiés	$O(r ^2 \times s)$	$O(r \times s)$
Plus non nullable	$O(2^{ r } \times s)$	$O(r \times s)$
Plus nullable greedy	$O(2^{ r } \times s)$	$O(r \times s)$
Lookaheads et Lookbehinds	non supporté	$O(r \times s)$

Le premier algorithme linéaire pour Lookaheads et Lookbehinds !

Grâce à la sémantique des groupes de capture JavaScript.

Des restrictions applicables à d'autres langages.

$|r|$: taille de la regex

$|s|$: taille de la chaîne de caractères

Comment raisonner sur les regex JavaScript?

Standard JS (pseudocode)

22.2.2.2 Runtime Semantics: CompilePattern

The syntax-directed operation `CompilePattern` takes argument `rer` (a `RegExp Record`) and returns an `Abstract Closure` that takes a `List` of characters and a non-negative `integer` and returns either a `MatchState` or `FAILURE`. It is defined piecewise over the following productions:

Pattern :: *Disjunction*

1. Let `m` be `CompileSubpattern` of `Disjunction` with arguments `rer` and `FORWARD`.
2. Return a new `Abstract Closure` with parameters `(Input, index)` that captures `rer` and `m` and performs the following steps when called:
 - a. `Assert`: `Input` is a `List` of characters.
 - b. `Assert`: $0 \leq index \leq$ the number of elements in `Input`.
 - c. Let `c` be a new `MatcherContinuation` with parameters `(y)` that captures nothing and performs the following steps when called:
 - i. `Assert`: `y` is a `MatchState`.
 - ii. Return `y`.
 - d. Let `cap` be a `List` of `rer.[[CapturingGroupsCount]]` **undefined** values, indexed 1 through `rer.[[CapturingGroupsCount]]`.
 - e. Let `x` be the `MatchState` { `[|Input|]: Input`, `[|EndIndex|]: index`, `[|Captures|]: cap` }.
 - f. Return `m(x, c)`.

Comment raisonner sur les regex JavaScript?

Standard JS (pseudocode)

2.2.2.2 Runtime Semantics: CompilePattern

The syntax-directed operation `CompilePattern` takes argument `rer` (a `RegExp Record`) and returns an `Abstract Closure` that takes a `List` of characters and a non-negative `integer` and returns either a `MatchState` or `FAILURE`. It is defined piecewise over the following productions:

`Pattern` :: `Disjunction`

1. Let `m` be `CompileSubpattern` of `Disjunction` with arguments `rer` and `FORWARD`.
2. Return a new `Abstract Closure` with parameters (`Input`, `index`) that captures `rer` and `m` and performs the following steps when called:
 - a. **Assert:** `Input` is a `List` of characters.
 - b. **Assert:** $0 \leq index \leq$ the number of elements in `Input`.
 - c. Let `c` be a new `MatcherContinuation` with parameters (`y`) that captures nothing and performs the following steps when called:
 - i. **Assert:** `y` is a `MatchState`.
 - ii. Return `y`.
 - d. Let `cap` be a `List` of `rer`.`[[CapturingGroupsCount]]` **undefined** values, indexed 1 through `rer`.`[[CapturingGroupsCount]]`.
 - e. Let `x` be the `MatchState` { `[[Input]]: Input`, `[[EndIndex]]: index`, `[[Captures]]: cap` }.
 - f. Return `m(x, c)`.

Modèles existants

Opération	t	Overapproximate Model for $(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t)$
Alternation	$t_1 \mid t_2$	$\{(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge C_{t-1} = \dots = C_{t-4} = \emptyset\}$ $\vee \{(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_2) \wedge C_1 = \dots = C_{t-1} = \emptyset\}$
Concaténation	$t_1 \cdot t_2$	$w = w_1 \cdots w_t \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w_2, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_2)$
Backreference-free	t_1^*	$w = w_1 \cdots w_t \wedge w_i \in \mathcal{L}(t[s]) \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1)[s]$ $\wedge \{w_1 = \dots = (w_1 = \epsilon \wedge C_1 = \dots = C_{t-1} = \emptyset)\}$
Positive Lookahead	$(?^t_1)t_2$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_{t+1}, \dots, C_{t+k}) \in \mathcal{L}_t(t_2)$
Negative Lookahead	$(?^t_1)^*t_2$	$(w, C_1, \dots, C_{t-1}) \notin \mathcal{L}_t(t_1) \wedge (w, C_{t+1}, \dots, C_{t+k}) \in \mathcal{L}_t(t_2)$
Input Start	t_1^+	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{t\}[\epsilon]$
Input Start (MultiLine)	t_1^*	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{t\}[\epsilon]$
Input End	t_1^l	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{t\}[\epsilon]$
Input End (MultiLine)	t_1^l	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{t\}[\epsilon]$ $w = w_1 \cdots w_t \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w_2, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_2)$ $\wedge \{\{w_1 \in \mathcal{L}(\epsilon, \star) \vee w_1 = \epsilon\} \wedge w_2 \in \mathcal{L}(\star, \epsilon) \} \vee \{w_1 \in \mathcal{L}(\star, \epsilon) \wedge (w_2 \in \mathcal{L}(\star, \epsilon) \vee w_2 = \epsilon)\}$
Word Boundary	$t_1(B) t_2$	$w = w_1 \cdots w_t \wedge w_1 \in \mathcal{L}(\star, \epsilon) \vee w_1 = \epsilon \wedge w_2 \in \mathcal{L}(\star, \epsilon) \wedge \{w_1 \in \mathcal{L}(\star, \epsilon) \wedge w_2 \in \mathcal{L}_t(t_2)\}$ $\wedge \{(w_1 \notin \mathcal{L}(\star, \epsilon)) \wedge w_1 = \epsilon\} \vee \{w_2 \notin \mathcal{L}(\star, \epsilon)\} \wedge \{w_1 \notin \mathcal{L}(\star, \epsilon) \wedge w_2 \in \mathcal{L}(\star, \epsilon)\}$
Non-Word Boundary	$t_1(B) t_2$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge C_1 = \epsilon$
Capture Group	$(?_1)$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1)$
Non-Capturing Group	$(?:_1)$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1)$
Base Case	$t \text{ regular}$	$w \in \mathcal{L}(t)$

Comment raisonner sur les regex JavaScript?

Standard JS (pseudocode)

22.2.2.2 Runtime Semantics: CompilePattern

The syntax-directed operation `CompilePattern` takes argument `rer` (a `RegExp Record`) and returns an `Abstract Closure` that takes a `List` of characters and a non-negative `integer` and returns either a `MatchState` or `FAILURE`. It is defined piecewise over the following productions:

`Pattern` :: `Disjunction`

1. Let `m` be `CompileSubpattern` of `Disjunction` with arguments `rer` and `FORWARD`.
2. Return a new `Abstract Closure` with parameters `(Input, index)` that captures `rer` and `m` and performs the following steps when called:
 - a. **Assert:** `Input` is a `List` of characters.
 - b. **Assert:** $0 \leq \text{index} \leq$ the number of elements in `Input`.
 - c. Let `c` be a new `MatcherContinuation` with parameters `(y)` that captures nothing and performs the following steps when called:
 - i. **Assert:** `y` is a `MatchState`.
 - ii. Return `y`.
 - d. Let `cap` be a `List` of `rer``[[CapturingGroupsCount]]` **undefined** values, indexed 1 through `rer``[[CapturingGroupsCount]]`.
 - e. Let `x` be the `MatchState` `{ [[Input]]; Input, [[EndIndex]]; index, [[Captures]]; cap }`.
 - f. Return `m(x, c)`.

Modèles existants

Opération	t	Overapproximate Model for $(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t)$
Alternation	$t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \notin \mathcal{L}_{t_1}(t_1) \wedge C_{t-1} = \dots = C_{t-1} = \emptyset \rangle$
		$\vee \langle (w, C_{t+1}, \dots, C_{t+1}) \in \mathcal{L}_{t_2}(t_2) \wedge C_t = \dots = C_{t-1} = \emptyset \rangle$
Concaténation	$t_1 \cdot t_2$	$w = w_1 \cdots w_t \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_1(t_1) \wedge (w_2, C_{t+1}, \dots, C_{t+1}) \in \mathcal{L}_2(t_2)$
Backreference-free	t_1^*	$w = w_1 \cdots w_t \wedge w_i \in \mathcal{L}(t_1)$ & $(w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_1(t_1)$
Quantification	t_1^+	$\wedge (w = \epsilon \iff (w_1 = \epsilon \wedge \dots \wedge C_{t-1} = \emptyset))$
Positive Lookahead	$(?^n)t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_{t_1}(t_1), w \rangle \wedge \langle (w, C_{t+1}, \dots, C_{t+1}) \in \mathcal{L}_2(t_2) \rangle$
Negative Lookahead	$(?^n)t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \notin \mathcal{L}_{t_1}(t_1), w \rangle \wedge \langle (w, C_{t+1}, \dots, C_{t+1}) \in \mathcal{L}_2(t_2) \rangle$
Empty Set	\emptyset	$w[p] = \emptyset$
	$p \leq w $	
	$w[p] = a$	
	$(a, w, p, \Lambda) \rightsquigarrow \langle (p+1, \Lambda) \rangle$	(CHARACTER)
	$p > w \vee w[p] \neq a$	$\langle (r, w, p, \Lambda) \rightsquigarrow \mathcal{N} \rangle$
	$(a, w, p, \Lambda) \rightsquigarrow \emptyset$	(CHARACTER FAILURE) $\langle (r, w, p, \Lambda) \rightsquigarrow \{(p_1, \Lambda_1) (p_1, \Lambda_1) \in \mathcal{N}\} \rangle$
	$(\emptyset, w, p, \Lambda) \rightsquigarrow \emptyset$	(CAPTURING GROUP)
	$(\emptyset, w, p, \Lambda) \rightsquigarrow \emptyset$	(EMPTY SET)
	$\langle (e, w, p, \Lambda) \rightsquigarrow \emptyset \rangle$	$\lambda(i) \neq \perp \wedge \langle (A(i), w, p, \Lambda) \rightsquigarrow \mathcal{N} \rangle$
	$\langle (e, w, p, \Lambda) \rightsquigarrow \{(p_1)\} \rangle$ (EMPTY STRING)	(\langle (v, w, p, \Lambda) \rightsquigarrow \mathcal{N} \rangle)
	$\langle (r_1, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \rangle \forall (p_1, \Lambda_1) \in \mathcal{N}, \langle r_2, w, p, \Lambda_1 \rangle \rightsquigarrow \mathcal{N}'$	$A(j) = \perp$
	$\langle (r_1r_2, w, p, \Lambda) \rightsquigarrow \bigcup_{p_1 \in \mathcal{N}} \mathcal{N}'_1 \rangle$ (CONCATENATION)	$\langle (v, w, p, \Lambda) \rightsquigarrow \emptyset \rangle$ (BACKREFERENCE FAILURE)
	$\langle (r_1, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \rangle \forall (p_1, \Lambda_1) \in \mathcal{N}$	$\langle (r, w, p, \Lambda) \rightsquigarrow \mathcal{N} \rangle$
	$\langle (r_1r_2, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \cup \mathcal{N}'' \rangle$ (UNION)	$\langle (\exists r), w, p, \Lambda \rangle \rightsquigarrow \{(p, A') (\exists r', w, p, \Lambda) \rightsquigarrow \mathcal{N}'\}$
	$\langle (r, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \rangle$	(POSITIVE LOOKAHEAD)
	$\langle (r, w, p, \Lambda) \rightsquigarrow \mathcal{N}'' \rangle$	$\langle (\exists r), w, p, \Lambda \rangle \rightsquigarrow \{(p, A') (\exists r', w, p, \Lambda) \rightsquigarrow \mathcal{N}''\}$
	$\forall (p_1, \Lambda_1) \in (\mathcal{N}' \setminus \{(p, \Lambda)\}), \langle r^*, w, p, \Lambda_1 \rangle \rightsquigarrow \mathcal{N}'_1$	(NEGATIVE LOOKAHEAD)
	$\langle (r^*, w, p, \Lambda) \rightsquigarrow \{(p, \Lambda)\} \cup \bigcup_{p_1 \in (\mathcal{N}' \setminus \{(p, \Lambda)\})} \mathcal{N}'_1 \rangle$ (REPETITION)	

Figure 2 Rules of the matching relation \rightsquigarrow .

Comment raisonner sur les regex JavaScript?

Problème : les modèles sémantiques existants sont incomplets ou faux.

Standard JS (pseudocode)

22.2.2.2 Runtime Semantics: CompilePattern

The syntax-directed operation `CompilePattern` takes argument `rer` (a RegExp Record) and returns an Abstract Closure that takes a List of characters and a non-negative integer and returns either a `MatchState` or `FAILURE`. It is defined piecewise over the following productions:

Pattern 2: Disjunction

1. Let m be `CompileSubpattern` of *Disjunction* with arguments *rer* and FORWARD.
 2. Return a new `Abstract Closure` with parameters (*Input*, *index*) that captures *rer* and *m* and performs the following steps when called:
 - a. Assert: *Input* is a List of characters.
 - b. Assert: $0 \leq \text{index} \leq$ the number of elements in *Input*.
 - c. Let t be a new `MatcherContinuation` with parameters (*y*) that captures nothing and performs the following steps when called:
 - i. Assert: *y* is a `MatchState`.
 - ii. Return *y*.
 - d. Let *cap* be a List of *rer*.`[[CapturingGroupsCount]]` **undefined** values, indexed 1 through *rer*.`[[CapturingGroupCount]]`.
 - e. Let *s* be the `MatchState` (`[[Input]]: Input, [[EndIndex]]: index, [[Captures]]: cap`).
 - f. Return *mfx*.*s*.

Non équivalent

Modèles existants

Operation	t	Overapproximate Model for $(w, C_1, \dots, C_k) \in \mathcal{L}_t(t)$
Alteration	$t_1 \sqcup t_2$	$\{w, (C_1, \dots, C_k) \in \mathcal{L}_t(t) \mid C_{i+k} = \emptyset\}$
Concatenation	$t_1 \cdot t_2$	$\{w, (C_1, \dots, C_{i-1}, C_i \cdot t_2) \in \mathcal{L}_t(t) \mid C_{i+k} = \emptyset\}$
Backreference-free Quantification	$t_{\exists x}$	$w = w \wedge \forall x \wedge \forall (w, C_1, \dots, C_k) \in \mathcal{L}_t(t) \mid \{w, (C_1, \dots, C_k) \in \mathcal{L}_t(t)\}$
Positive Lookahead	$(\exists t_2) t_1$	$\{w, (C_1, \dots, C_k) \in \mathcal{L}_t(t_1) \mid \{w, (C_1, \dots, C_k) \in \mathcal{L}_t(t_2)\}$
Negative Lookahead	$(\nexists t_2) t_1$	$\{w, (C_1, \dots, C_k) \notin \mathcal{L}_t(t_1) \mid \{w, (C_1, \dots, C_k) \in \mathcal{L}_t(t_2)\}$
<hr/>		
Empty Set		$\{\emptyset\}$
	$D \leq w $	$w[\vec{p}] = \emptyset$
	$(w, v, p, \lambda) \rightarrow \{\emptyset\}$	(CHARACTER)
1	$p \geq v \vee w[\vec{p}] \neq \emptyset$	$(w, v, p, \lambda) \rightarrow \mathcal{N}$
2	$(w, v, p, \lambda) \rightarrow \emptyset$	(CHARACTER FAILURE)
3	$(\emptyset, \emptyset, \emptyset, \emptyset) \rightarrow \emptyset$	(CAPTURING GS)
4	$(w, v, p, \lambda) \rightarrow \emptyset$	(EMPTY SET)
5	$(w, v, p, \lambda) \rightarrow \{\emptyset\}$	(EMPTY STRING)
6	$(r_1, w, p, \lambda) \rightarrow N_{(w, p, \lambda)}$	($N_{(w, p, \lambda)}$)
7	$(r_1, w, p, \lambda) \rightarrow \bigcup_{N \in \mathcal{N}_{(w, p, \lambda)}} N$	($\bigcup_{N \in \mathcal{N}_{(w, p, \lambda)}} N$)
8	$(r_1, r_2, w, p, \lambda) \rightarrow \bigcup_{N \in \mathcal{N}_{(w, p, \lambda)}} N$	(CONCATENATION)
9	$(r_1, w, p, \lambda) \rightarrow \mathcal{N}'$	$(r_1, w, p, \lambda) \rightarrow \mathcal{N}'$
10	$(r_1, w, p, \lambda) \rightarrow N_{(w, p, \lambda)} \cup N_{(r_1, w, p, \lambda)}$	(UNION)
11	$(r_1, w, p, \lambda) \rightarrow \mathcal{N}$	(\mathcal{N})
12	$\forall (p_1, p_2) \in \{(\vec{p}, \vec{q}), (\vec{p}, \vec{r})\}, (w, p_1, p_2, \lambda) \rightarrow N_{(w, p_1, p_2, \lambda)}$	($\forall (p_1, p_2) \in \{(\vec{p}, \vec{q}), (\vec{p}, \vec{r})\}, (w, p_1, p_2, \lambda) \rightarrow N_{(w, p_1, p_2, \lambda)}$)
13	$(r^*, w, p, \lambda) \rightarrow \{(\vec{p}, \vec{q})\} \cup \bigcup_{1 \leq i \leq k} G_i(w, p, \lambda) \cap N_{(w, p, \lambda)}$	(REPETITION)
14	$(r^*, w, p, \lambda) \rightarrow \mathcal{N}$	(\mathcal{N})
15	$(r^*, w, p, \lambda) \rightarrow \{(\vec{p}, \vec{q})\}$	($\{(\vec{p}, \vec{q})\}$)
16	$(r^*, w, p, \lambda) \rightarrow \mathcal{N}'$	(\mathcal{N}')
17	$(r^*, w, p, \lambda) \rightarrow \mathcal{N}' \wedge \text{str}(w \neq \emptyset, [p, \vec{p}])$	(POSITIVE LOOKARHEAD)
18	$(r^*, w, p, \lambda) \rightarrow \mathcal{N}' \wedge \text{str}(w = \emptyset, [p, \vec{p}])$	(NEGATIVE LOOKARHEAD)

■ Figure 2 Rules of the matching relation \sim

Comment raisonner sur les regex JavaScript?

Problème : les modèles sémantiques existants sont incomplets ou faux.

Standard JS (pseudocode)

22.2.2.2 Runtime Semantics: CompilePattern

The syntax-directed operation `CompilePattern` takes argument `rer` (a `RegExp Record`) and returns an `Abstract Closure` that takes a `List` of characters and a non-negative `integer` and returns either a `MatchState` or `FAILURE`. It is defined piecewise over the following productions:

Pattern :: *Disjunction*

1. Let `m` be `CompileSubpattern` of `Disjunction` with arguments `rer` and `FORWARD`.
2. Return a new `Abstract Closure` with parameters (`Input`, `index`) that captures `rer` and `m` and performs the following steps when called:
 - a. **Assert:** `Input` is a `List` of characters.
 - b. **Assert:** $0 \leq index \leq$ the number of elements in `Input`.
 - c. Let `c` be a new `MatcherContinuation` with parameters (`y`) that captures nothing and performs the following steps when called:
 - i. **Assert:** `y` is a `MatchState`.
 - ii. Return `y`.
 - d. Let `cap` be a `List` of `rer.[[CapturingGroupsCount]]` `undefined` values, indexed 1 through `rer.[[CapturingGroupsCount]]`.
 - e. Let `x` be the `MatchState` { `[[Input]]: Input`, `[[EndIndex]]: index`, `[[Captures]]: cap` }.
 - f. Return `m(x, c)`.

Équivalent

Nouvelle Mécanisation (Coq)

```
(*> Disjunction : Alternative | Disjunction <*)
Disjunction r1 r2 =>
(*> 1. Let m1 be CompileSubpattern of Alternative with arguments rer and direction. <*
let! m1 ==> compileSubPattern r1 (Disjunction_left r2 :: ctx) rer direction in
(*> 2. Let m2 be CompileSubpattern of Disjunction with arguments rer and direction. <*
let! m2 ==> compileSubPattern r2 (Disjunction_right r1 :: ctx) rer direction in
(*> 3. Return a new Matcher with parameters (x, c) that captures m1 and m2 and performs
the following steps when called: <*)
(λ (x: MatchState) (c: MatcherContinuation) =>
(*> a. Assert x is a MatchState. <*)
(*> b. Assert c is a MatcherContinuation. <*)
(*> c. Let r be m(x, c). <*)
let! r ==> m1 x c in
(*> d. If r is not failure, return r. <*)
if r is not failure then r
(*> e. Return m2(x, c). <*)
else m2 x c): Matcher
```

Une sémantique mécanisée de confiance

Thèse de master encadrée : Mécanisation du chapitre regex du standard JavaScript en Coq/Rocq.
🏅 compétition étudiante de PLDI.

Nouveaux algorithmes

Mes algorithmes sont intégrés dans V8 (2500 lignes de C++) :



Nouvelle sémantique

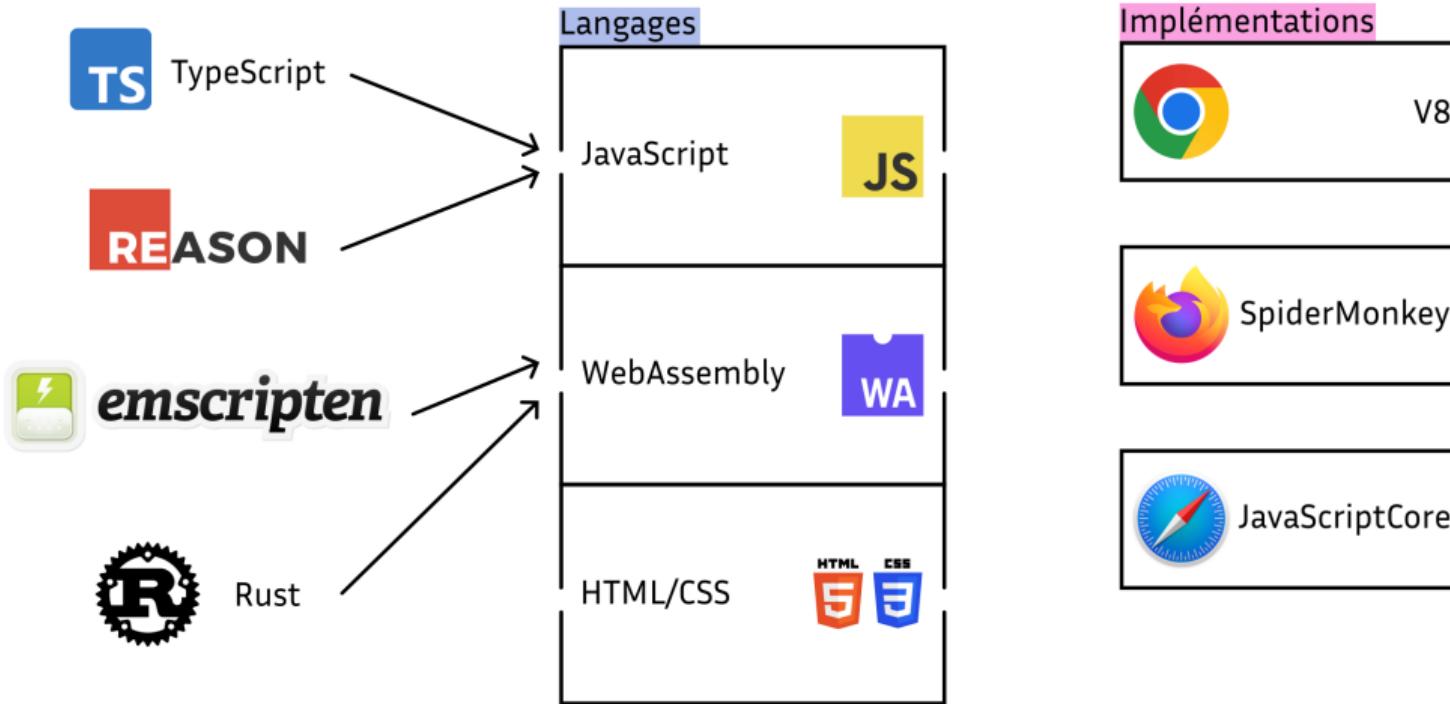
On peut enfin vérifier formellement des moteurs de regex JavaScript !

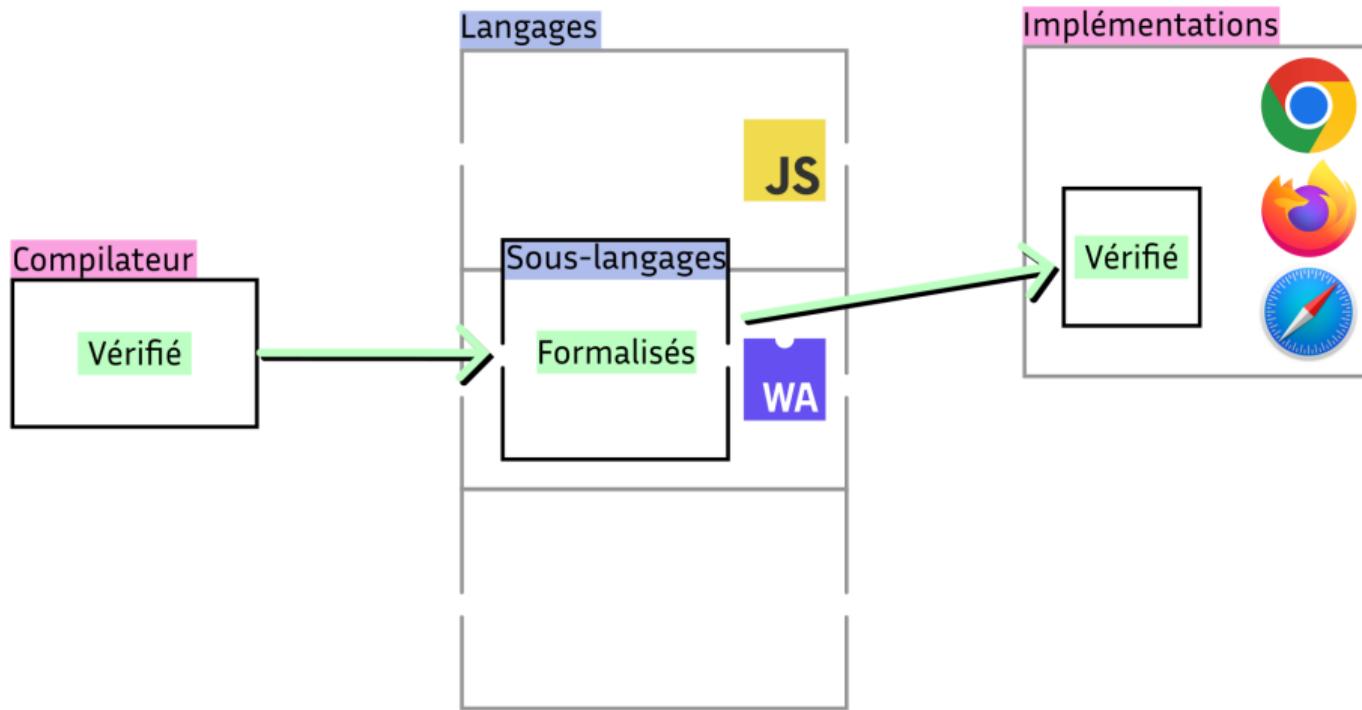
Publications : [PLDI'24], [ICFP'24]
Encadrement de 12 projets d'étudiants.

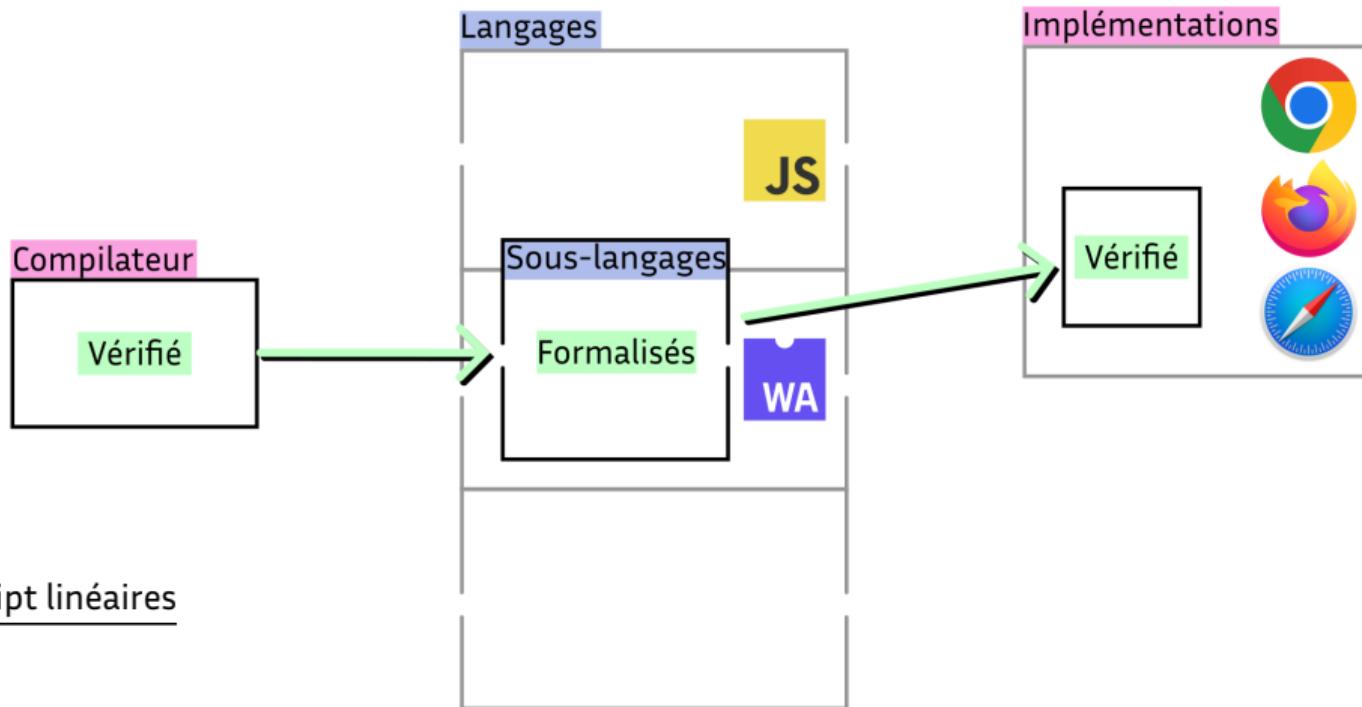
Financements :
Open Research Data Contribute Grant (32 000€)
Swiss National Science Foundation Project (497 000€)

Programme de Recherche

Vers une plateforme web de confiance : vérification formelle de compilateurs et d'environnements d'exécution



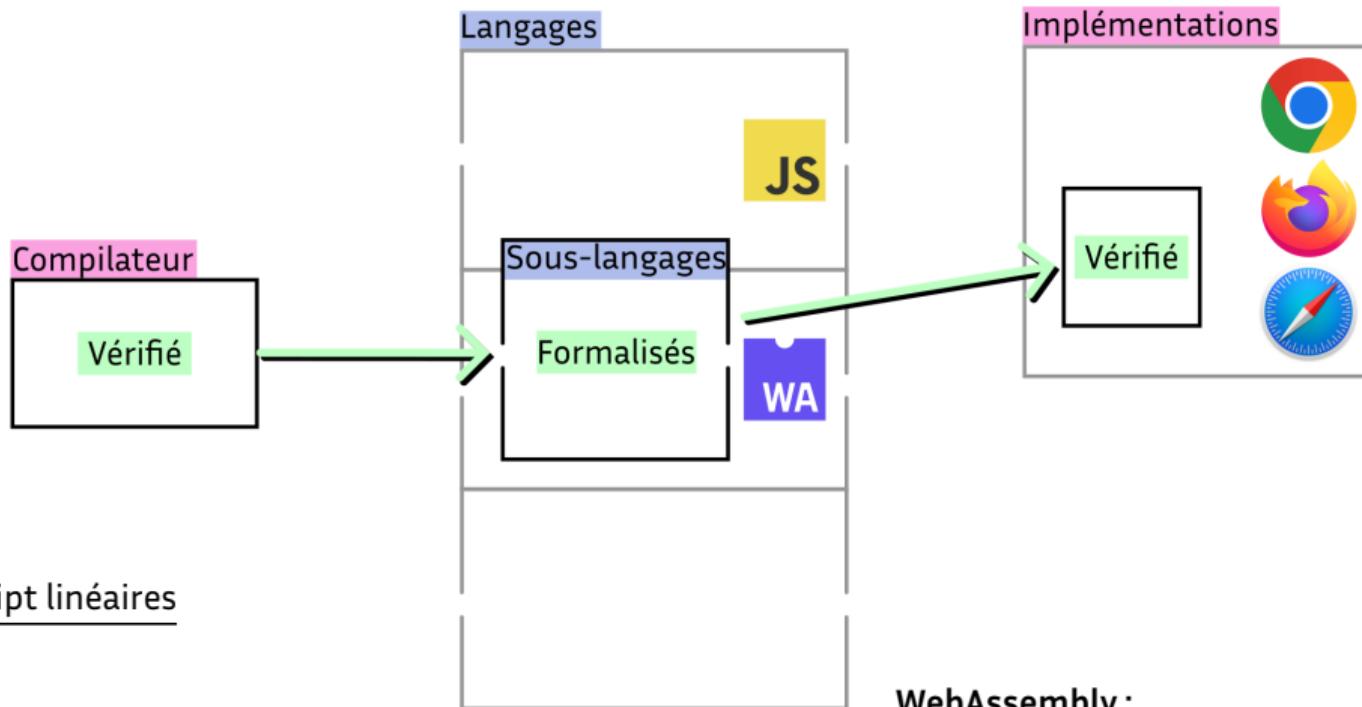




Deux axes :

Regex JavaScript linéaires

Un sous-ensemble de WebAssembly



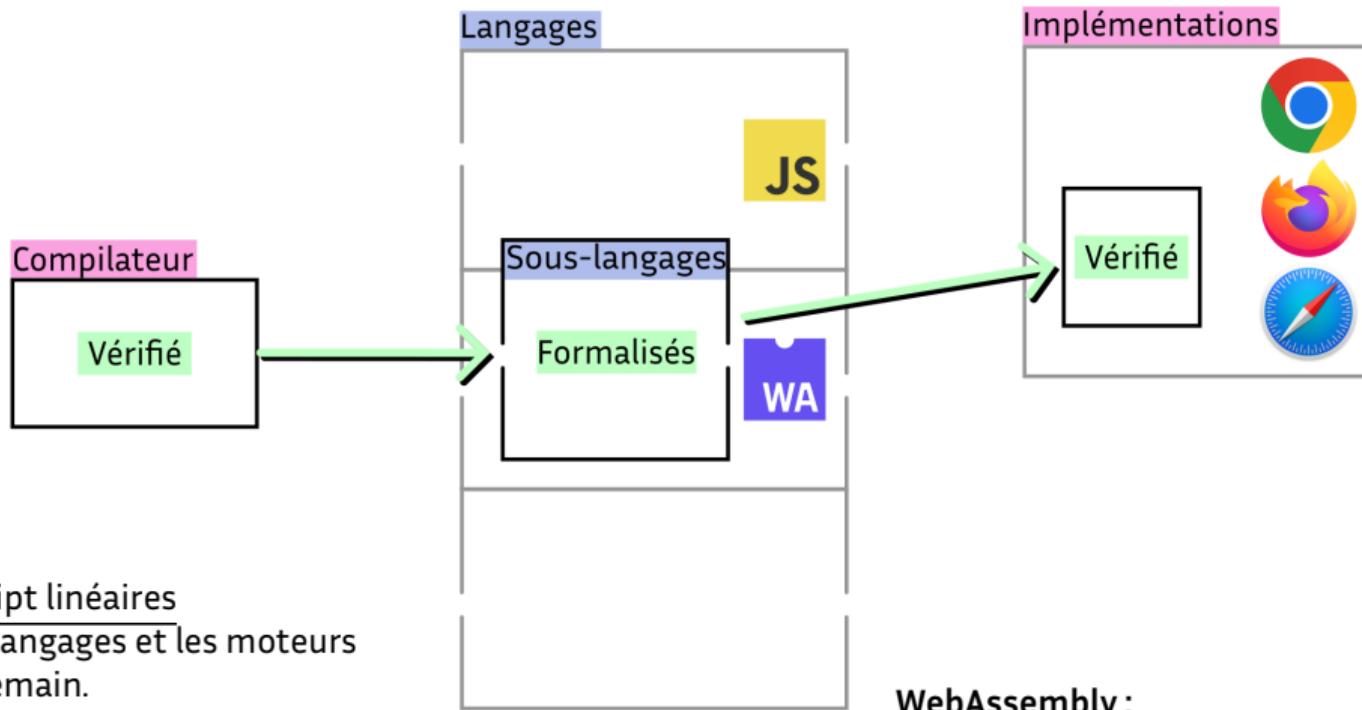
Deux axes :

Regex JavaScript linéaires

Un sous-ensemble de WebAssembly

WebAssembly :

- Bytecode bas niveau.
- Sémantique isolant chaque module.
- Avec une sémantique formelle.



Deux axes :

Regex JavaScript linéaires

Concevoir les langages et les moteurs de regex de demain.

Un sous-ensemble de WebAssembly

Compilation formellement vérifiée pour des programmes compartimentés.

WebAssembly :

- Bytecode bas niveau.
- Sémantique isolant chaque module.
- Avec une sémantique formelle.

Regex JavaScript

Sous-ensemble linéaire

JS

Regex JavaScript

Sous-ensemble linéaire

Lookaheads

Lookbehinds

Groupes de capture

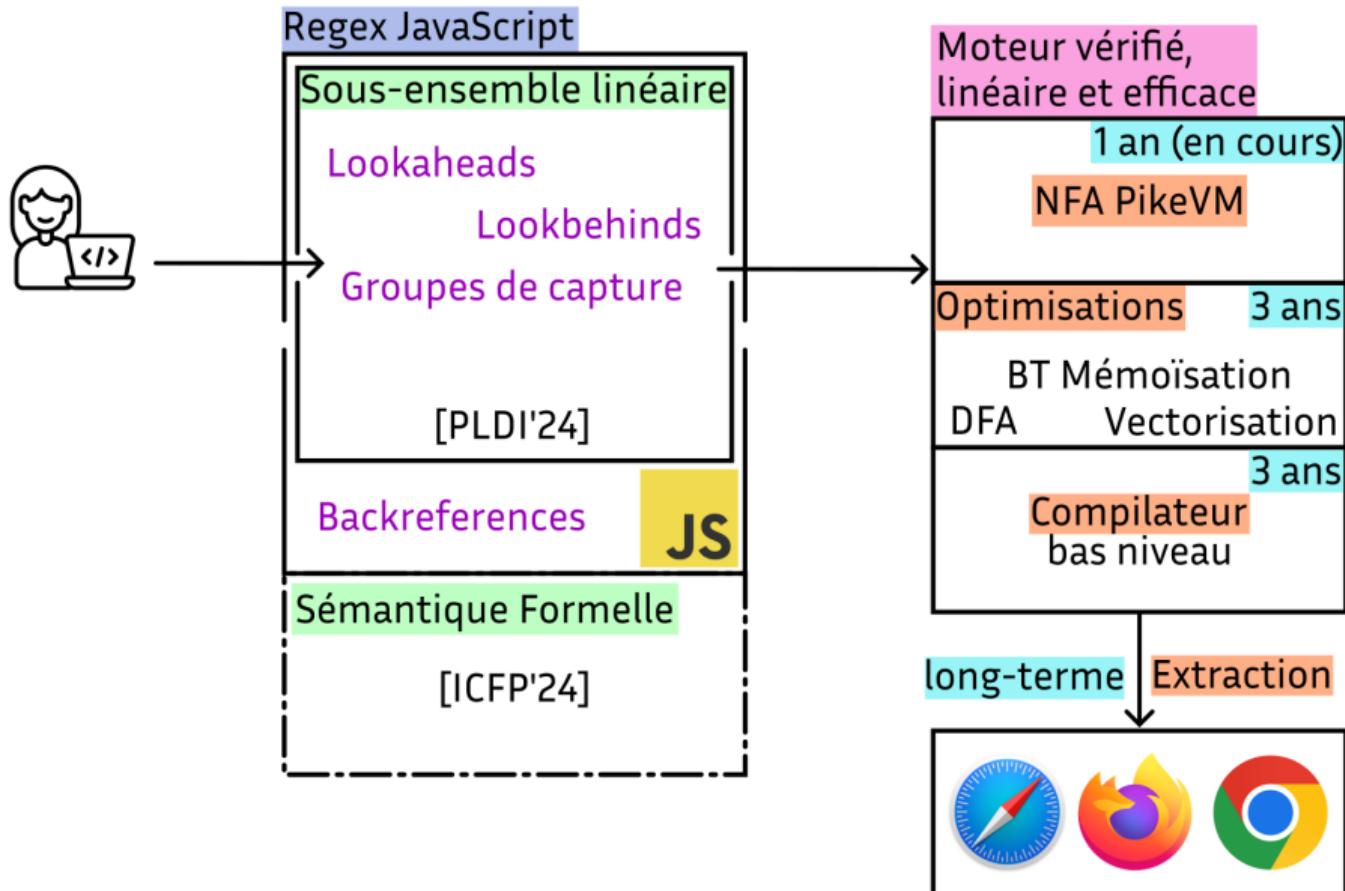
[PLDI'24]

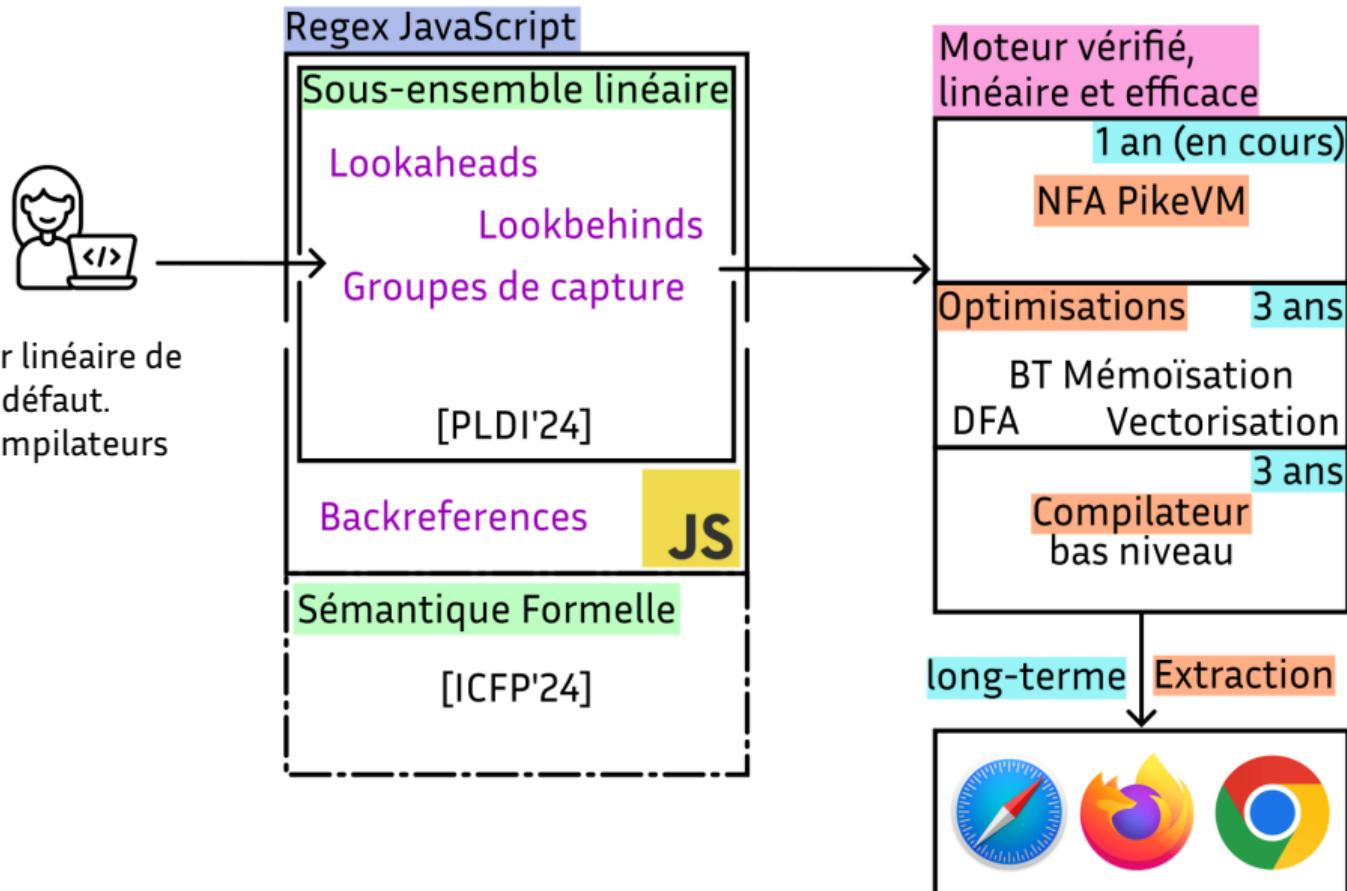
Backreferences

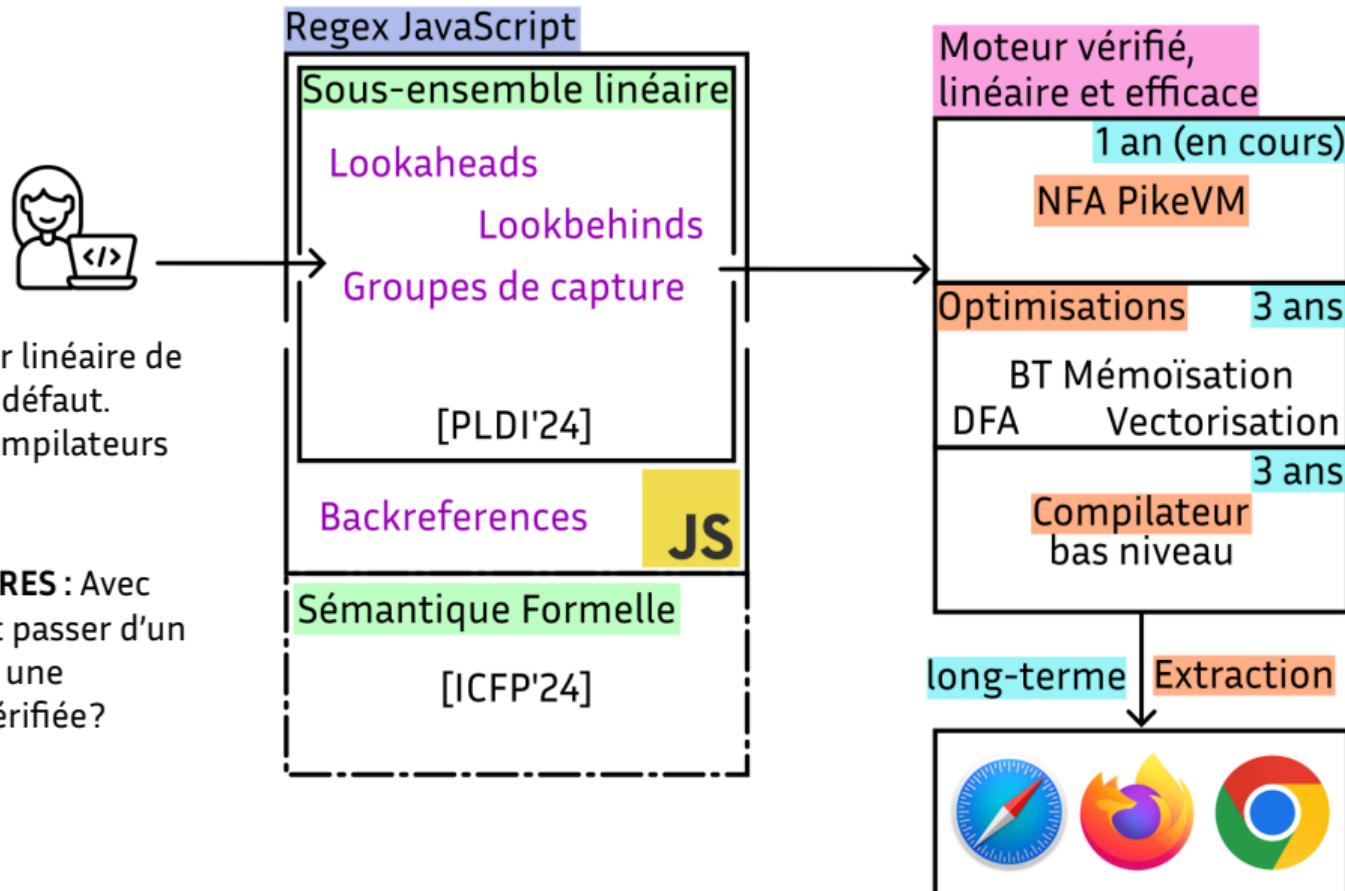
JS

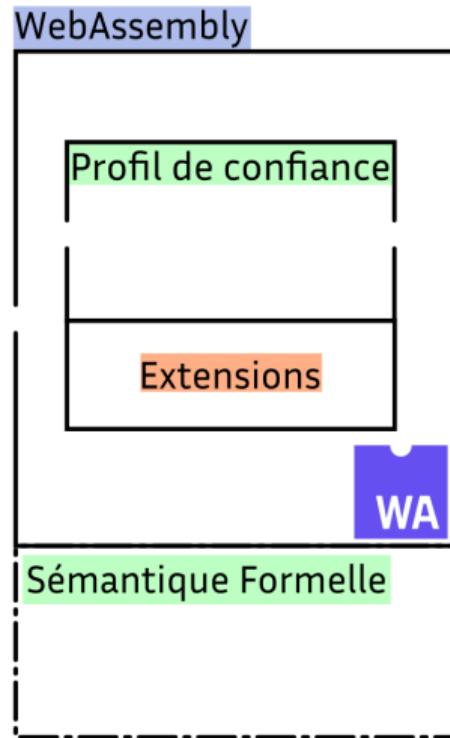
Sémantique Formelle

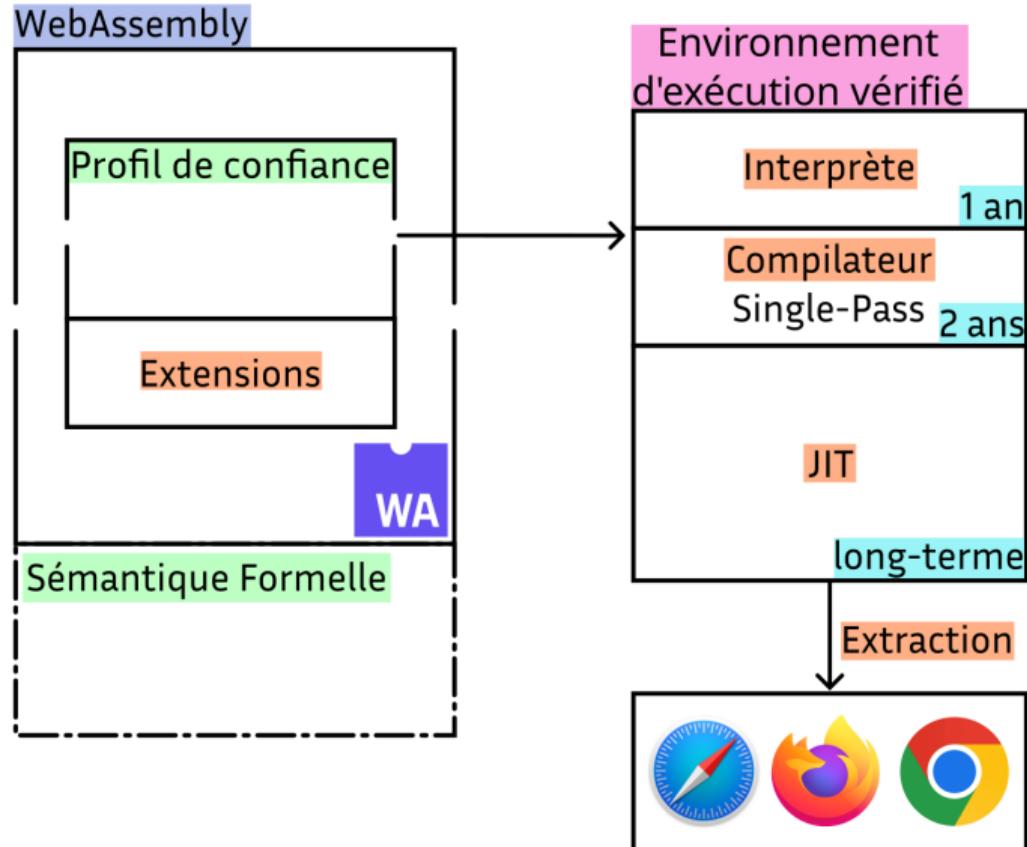
[ICFP'24]

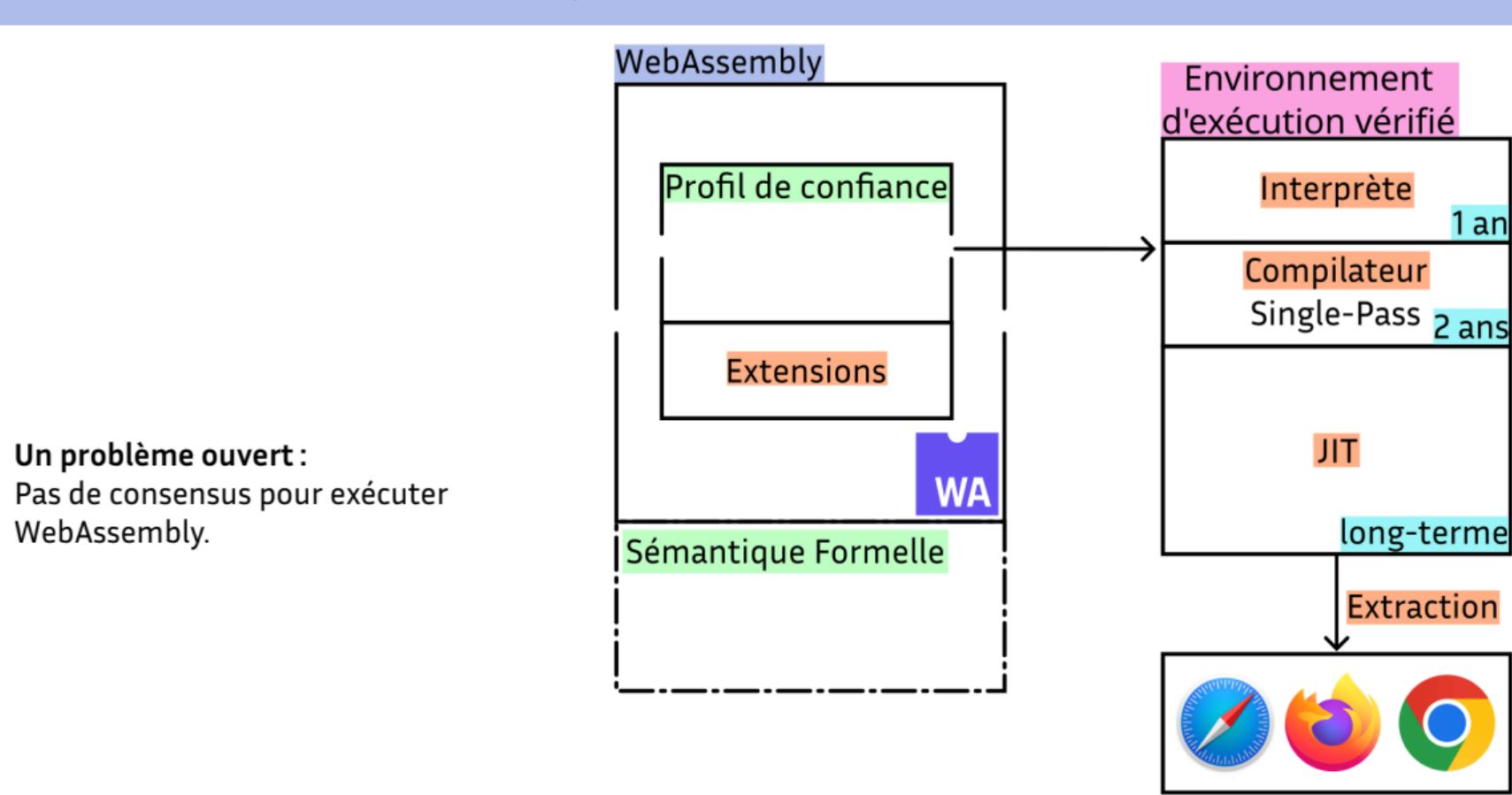


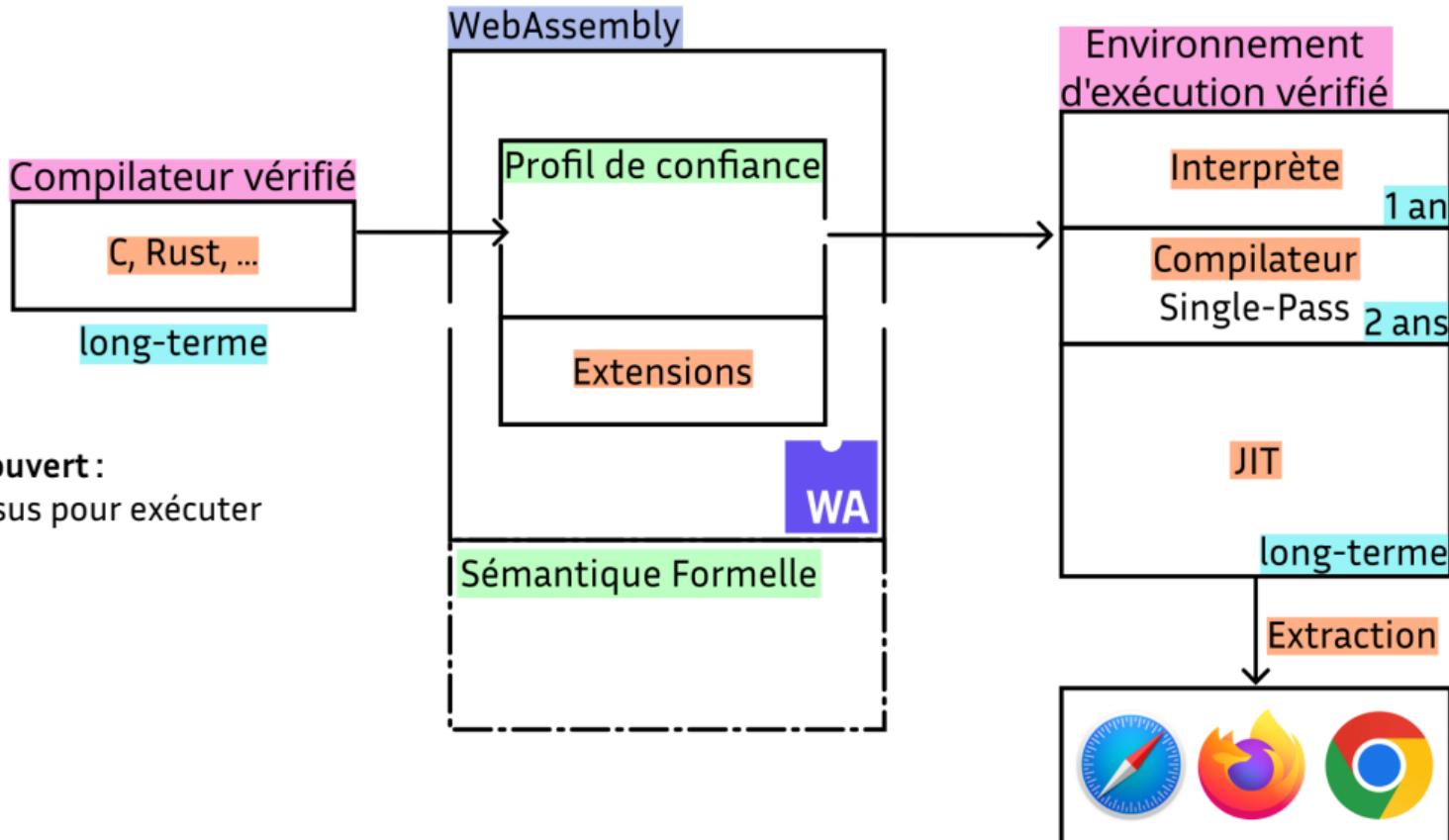


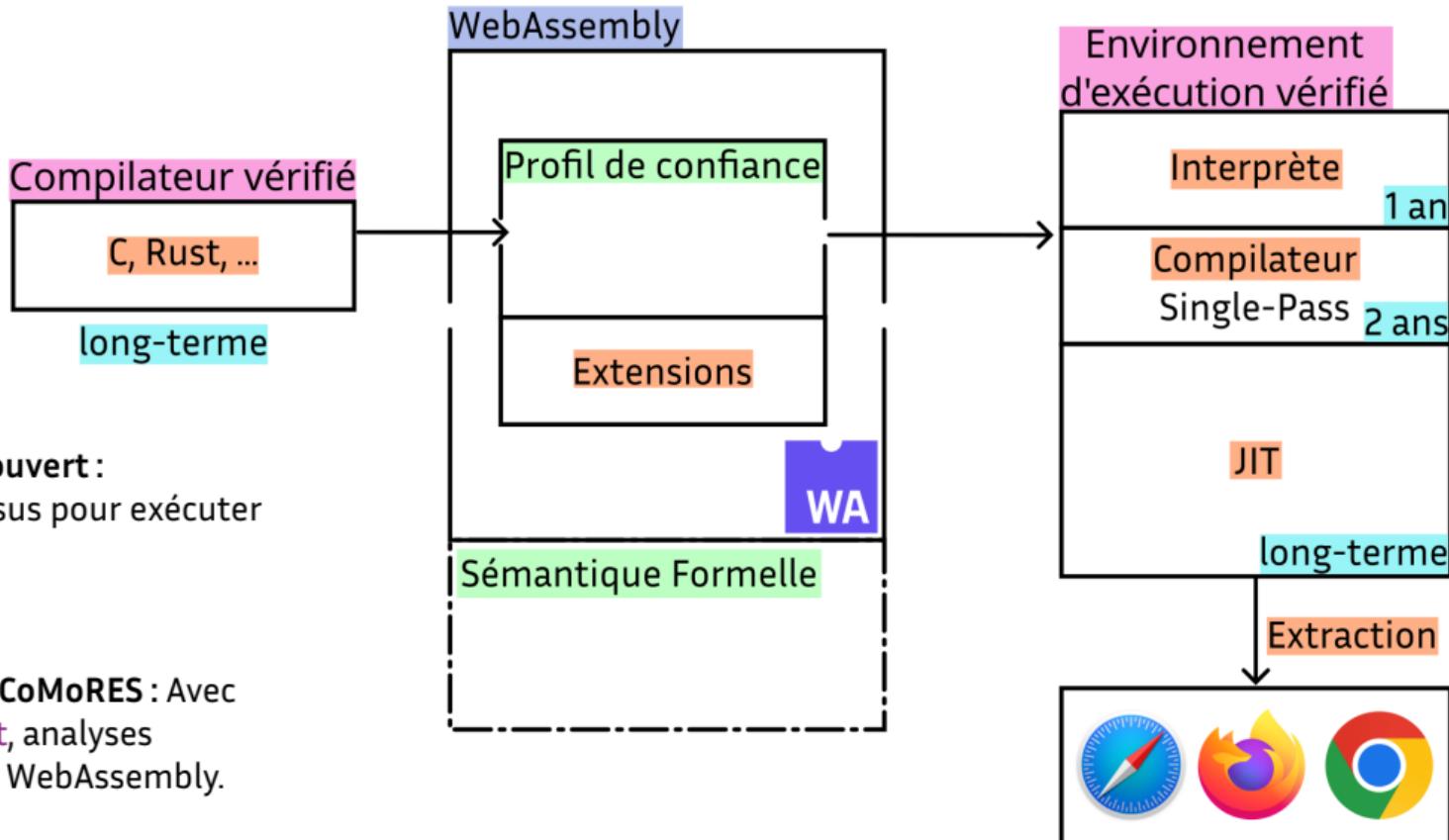












SyCoMoRES, Thèmes Communs

Méthodes formelles pour la vérification

Patrick Baillot, Raphaël Monat, Vlad Rusu

Vérification formelle en Coq/Rocq

Vlad Rusu

Sémantique de langages dynamiques

Raphaël Monat

Sémantique multi-langages

Raphaël Monat

Nouvelle expertise apportée

JITs, Regex, Compilation formellement vérifiée.

SyCoMoRES, Thèmes Communs

Méthodes formelles pour la vérification

Patrick Baillot, Raphaël Monat, Vlad Rusu

Vérification formelle en Coq/Rocq

Vlad Rusu

Sémantique de langages dynamiques

Raphaël Monat

Sémantique multi-langages

Raphaël Monat

Nouvelle expertise apportée

JITs, Regex, Compilation formellement vérifiée.

Troisième axe : compilation vérifiée pour systèmes temps-réel

Julien Forget et Giuseppe Lipari : systèmes temps-réel.

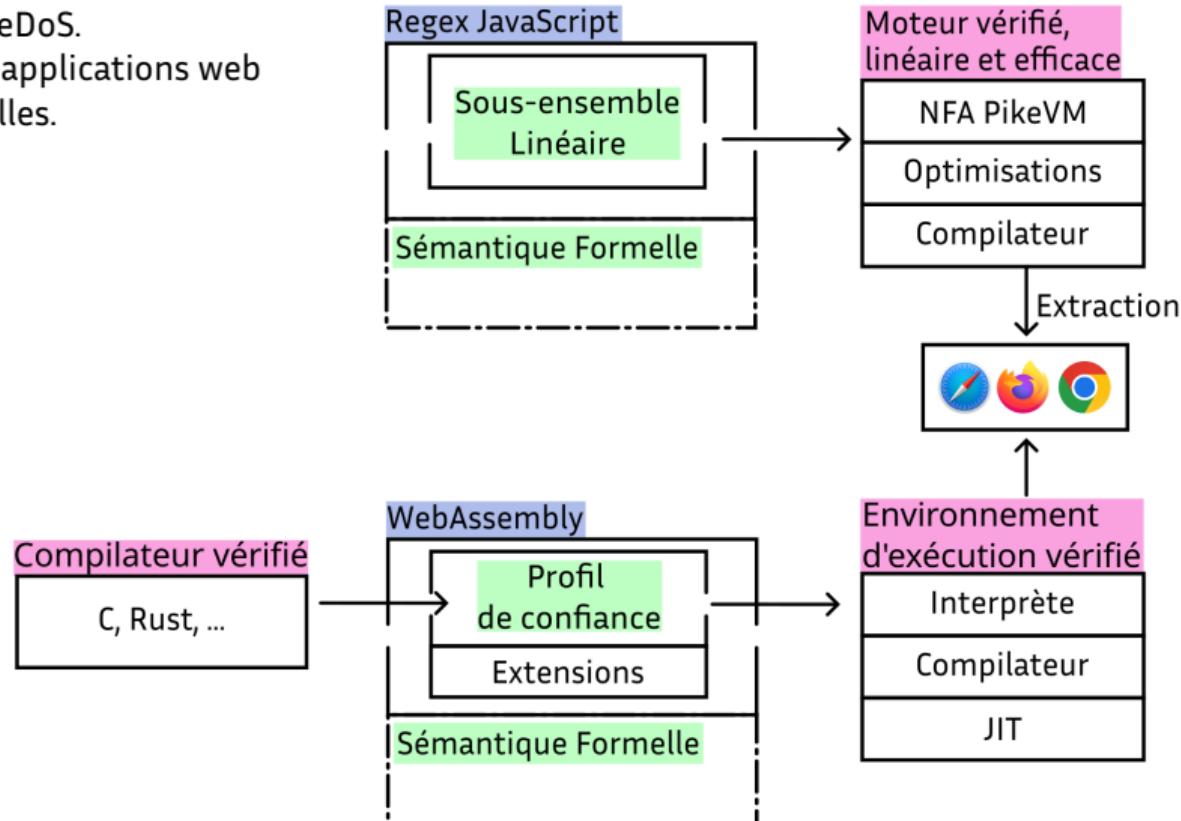
Objectif long-terme de l'équipe : compilateur formallement vérifié pour Prelude (langage dataflow synchrone avec contraintes temps-réel).

Comment garantir que le code compilé produit les bonnes valeurs aux dates attendues ?

Programme de recherche :

Vaincre la vulnérabilité ReDoS.

Déployer et exécuter des applications web avec des garanties formelles.



Programme de recherche :

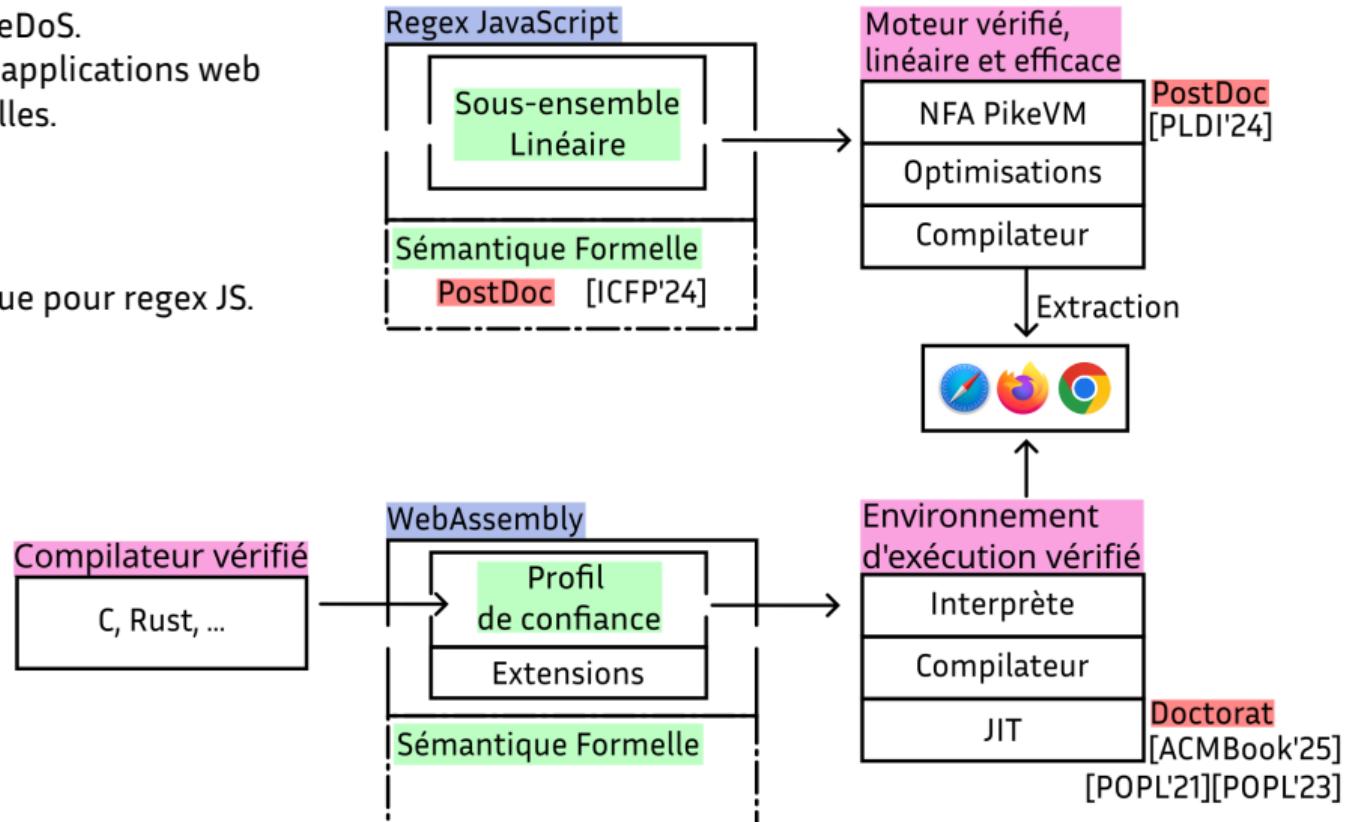
Vaincre la vulnérabilité ReDoS.

Déployer et exécuter des applications web avec des garanties formelles.

Travaux précédents :

JITs vérifiés.

Algorithmes et sémantique pour regex JS.



Programme de recherche :

Vaincre la vulnérabilité ReDoS.

Déployer et exécuter des applications web avec des garanties formelles.

Travaux précédents :

JITs vérifiés.

Algorithmes et sémantique pour regex JS.

Intégration SyCoMoRES :

Renforcer la thématique de vérification.

Apporter mon expertise en compilation vérifiée.

