

Aurèle Barrière

Doctorat, IRISA 

avec Sandrine Blazy et David Pichardie

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

2019-2022

PostDoc, EPFL 

avec Clément Pit-Claudel

*Vers des moteurs de regex modernes
linéaires et vérifiés*

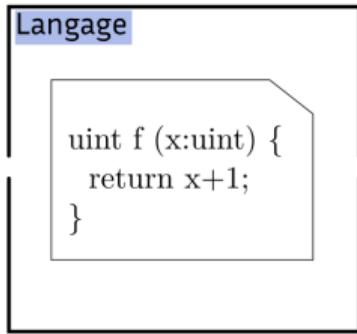
2023-2025

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

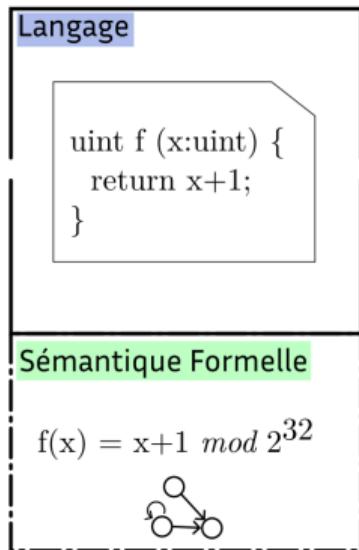
Propositions d'affectation : Cristal (Lille), LIP (Lyon), Verimag (Grenoble).

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

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

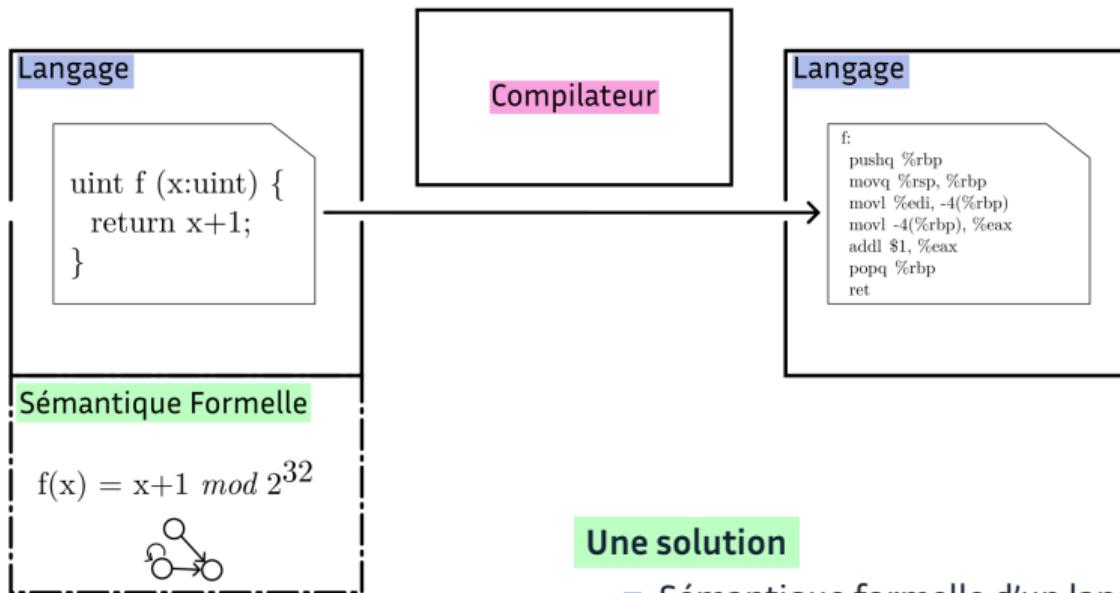


Comment faire confiance à l'exécution d'un programme ?

**Une solution**

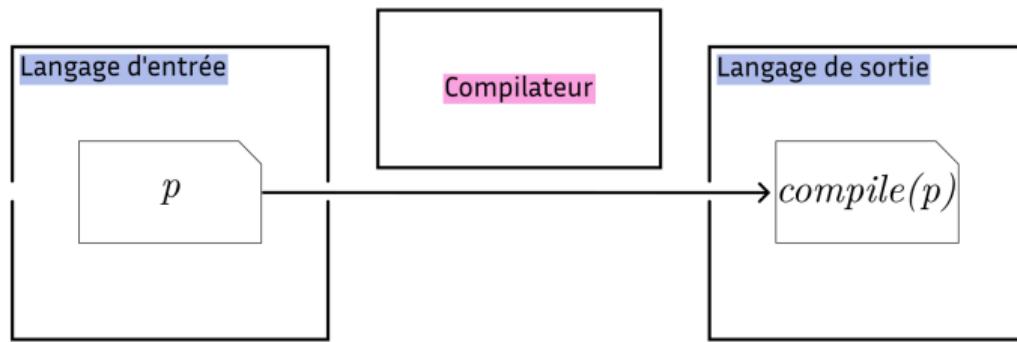
- Sémantique formelle d'un langage

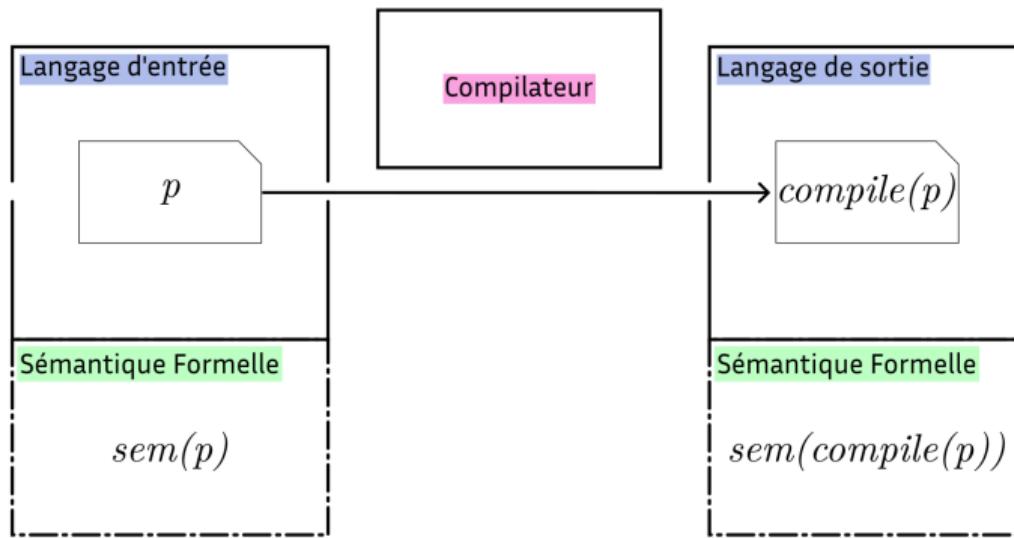
Comment faire confiance à l'exécution d'un programme ?

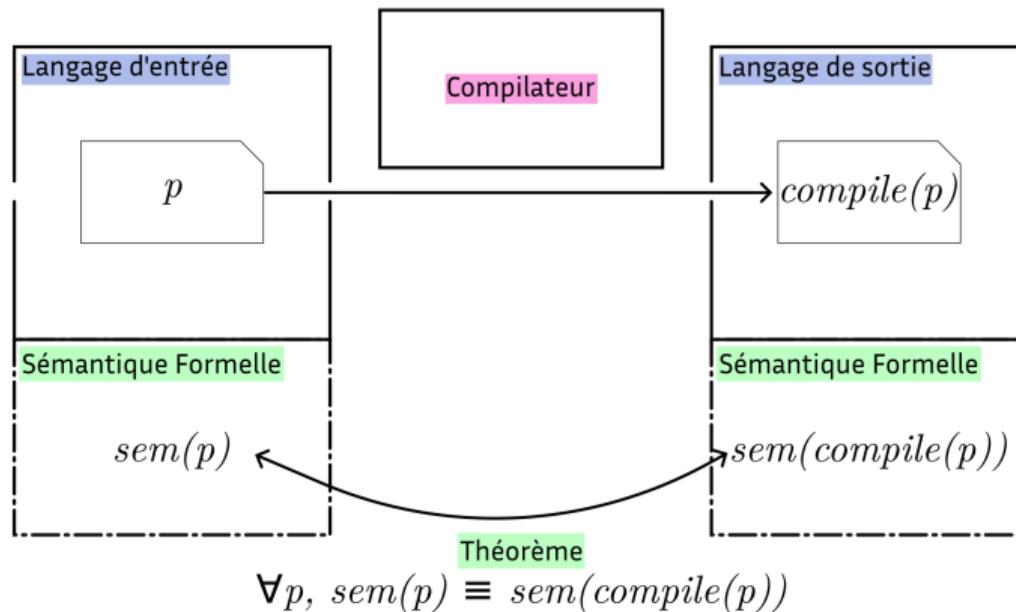


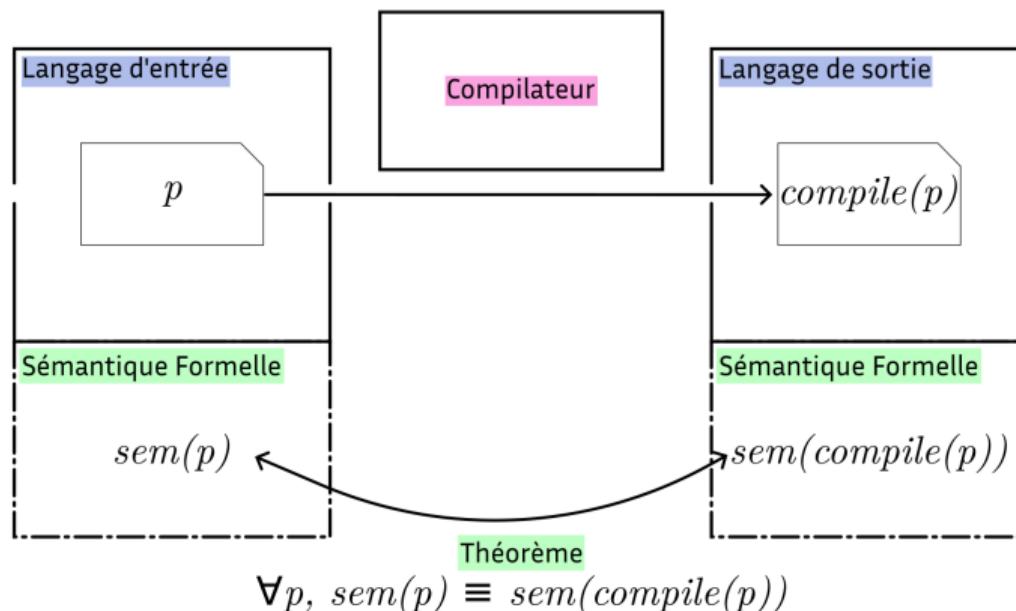
Une solution

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









Compilateurs formellement vérifiés

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?

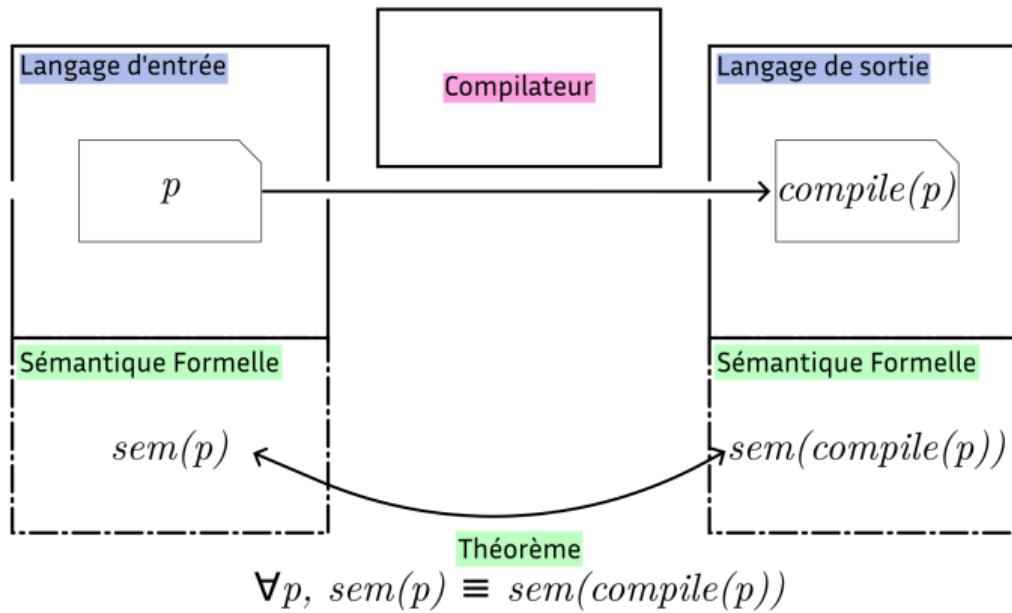
Un besoin de garanties

- 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.

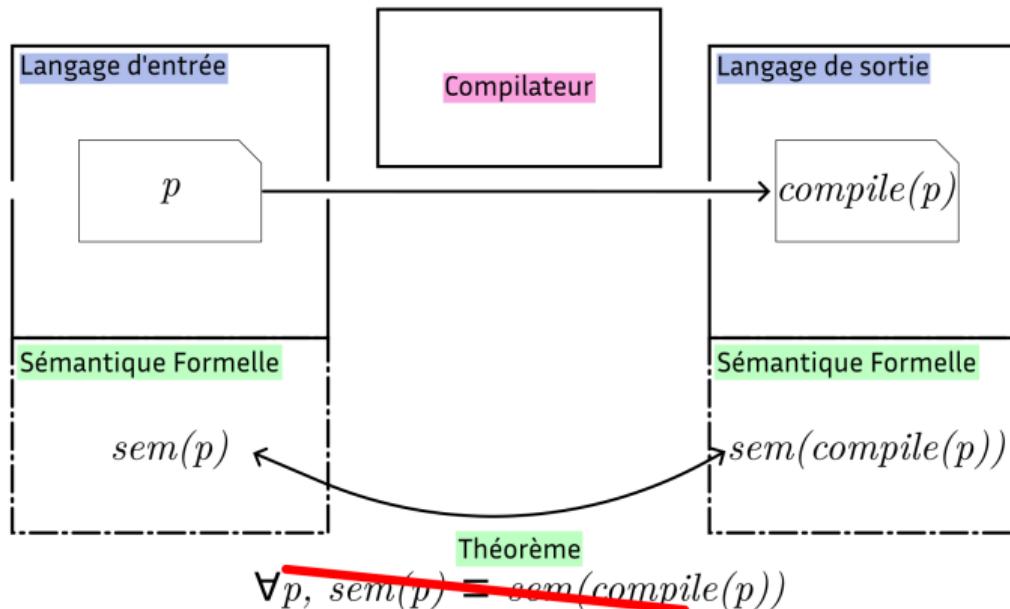
Comment faire confiance à l'exécution d'un programme sur le web?



Un problème

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

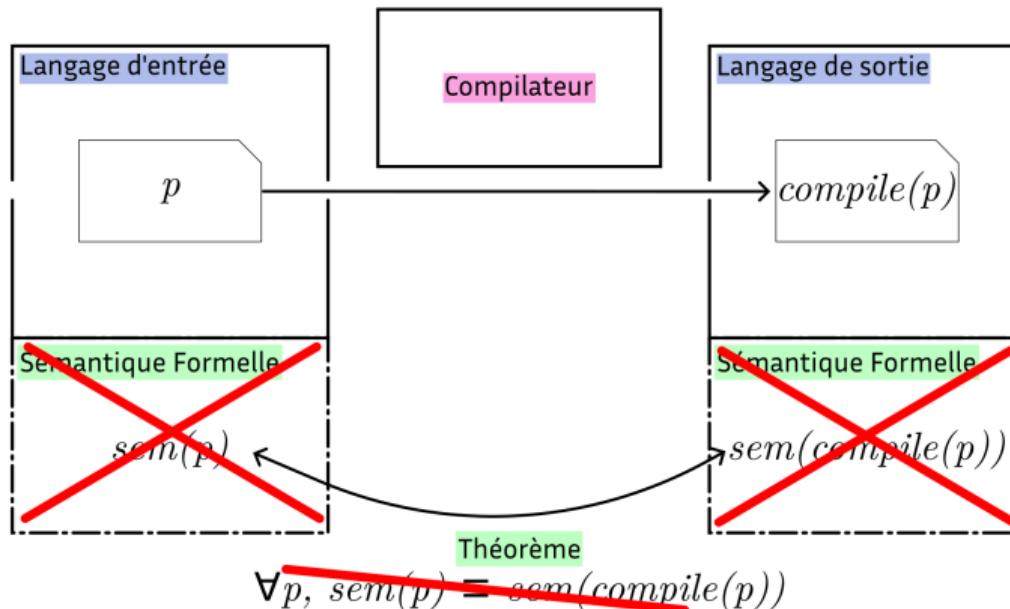
Comment faire confiance à l'exécution d'un programme sur le web?



Un problème

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

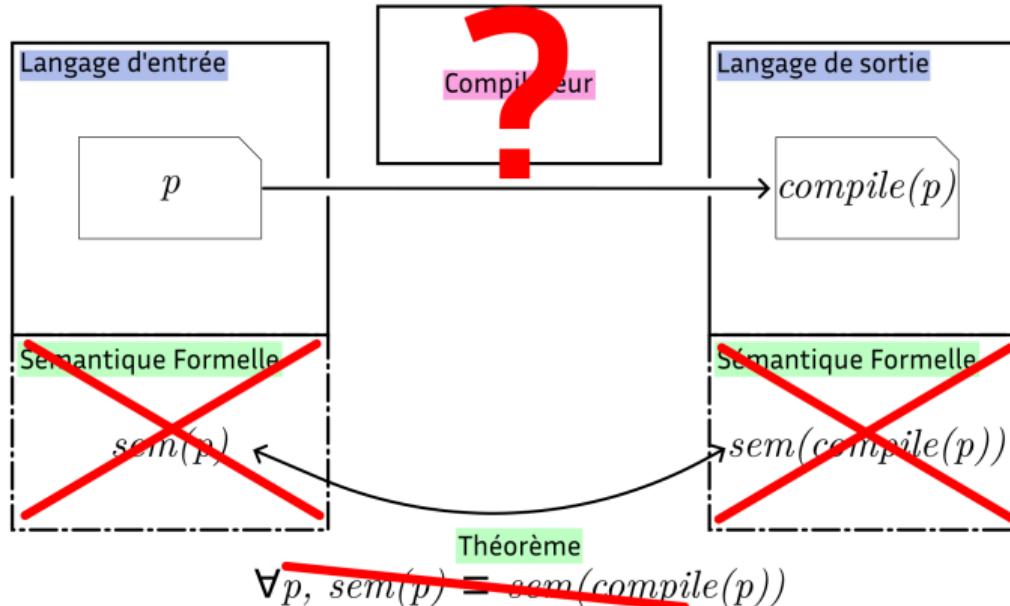
Comment faire confiance à l'exécution d'un programme sur le web?



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

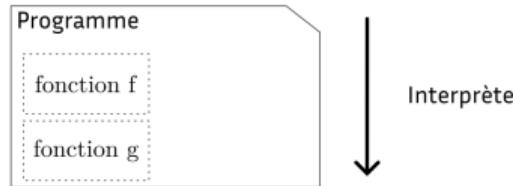
JIT (Just-in-Time) = entremêler **exécution et compilation** du programme.

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.

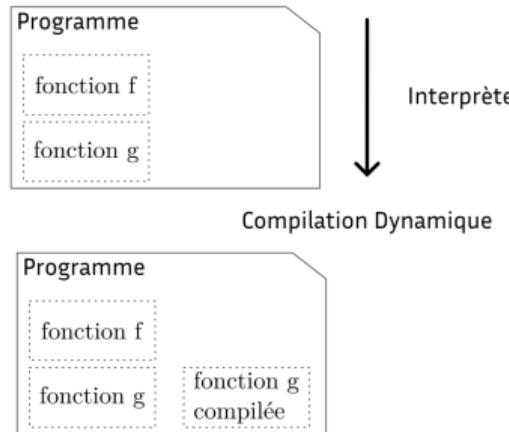


Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

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.

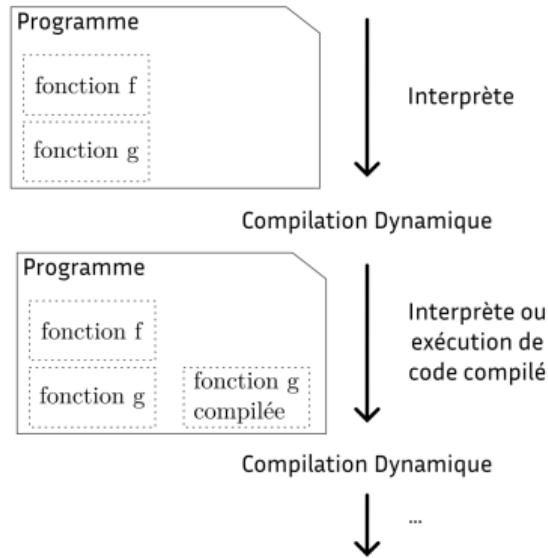


Publications : [ACMBooks'25], [POPL'23], [POPL'21], [CoqPL'20].

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

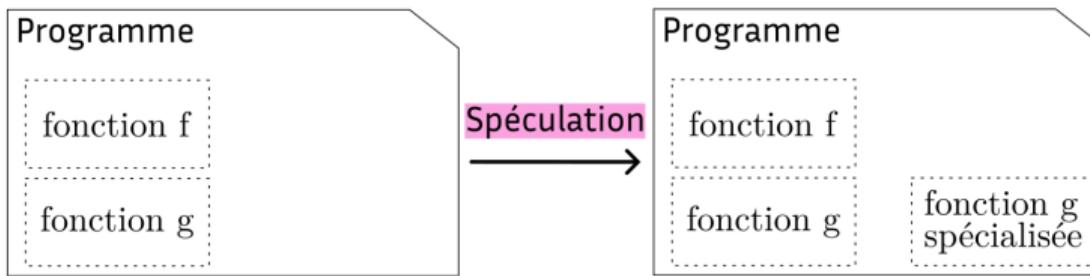
Un JIT génère du code exécutable ! 2019 : Coinbase est victime d'une attaque.

Comment écrire un compilateur JIT correct ?

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

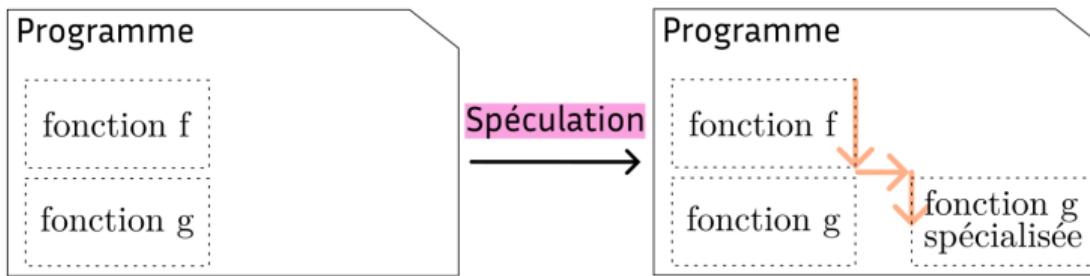
Prix de thèse : 🏆 EAPLS Best PhD Dissertation Award.

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



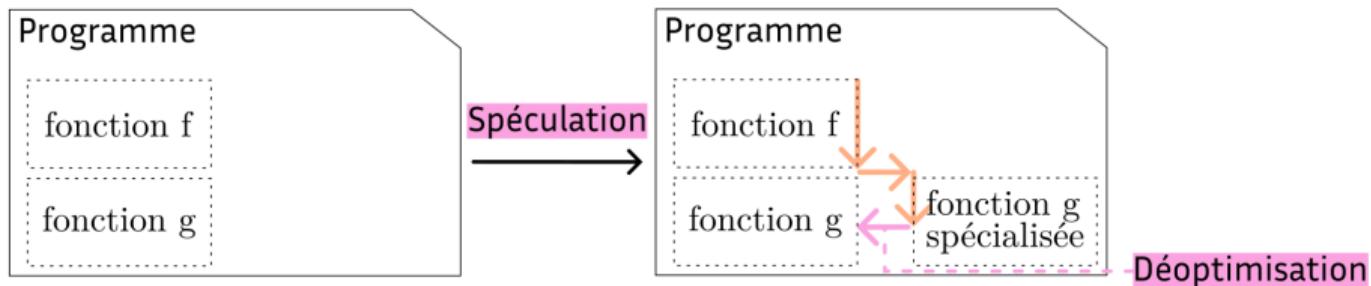
Spéculation

Compiler des versions spécialisées de fonctions.



Spéculation

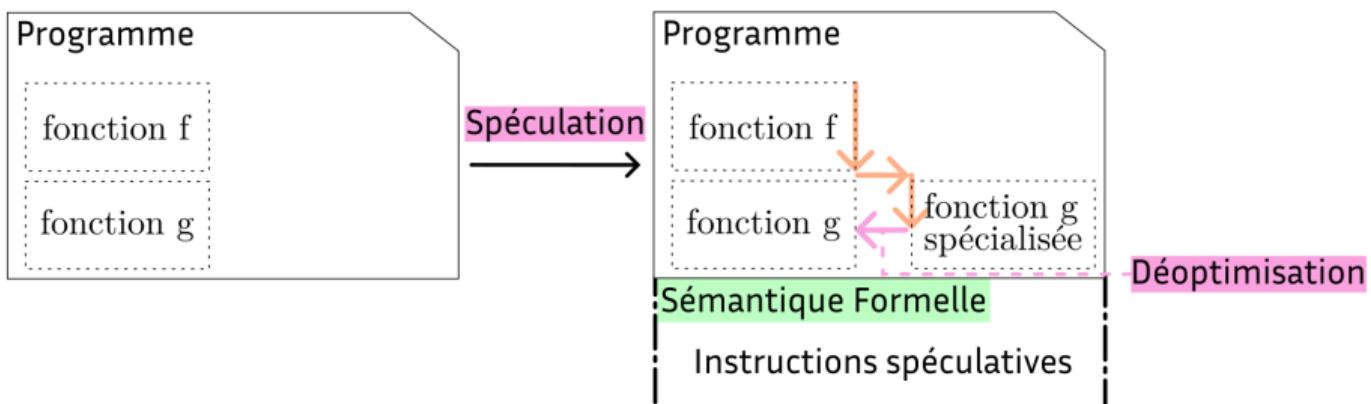
Compiler des versions spécialisées de fonctions.



Spéculation

Compiler des versions spécialisées de fonctions.

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



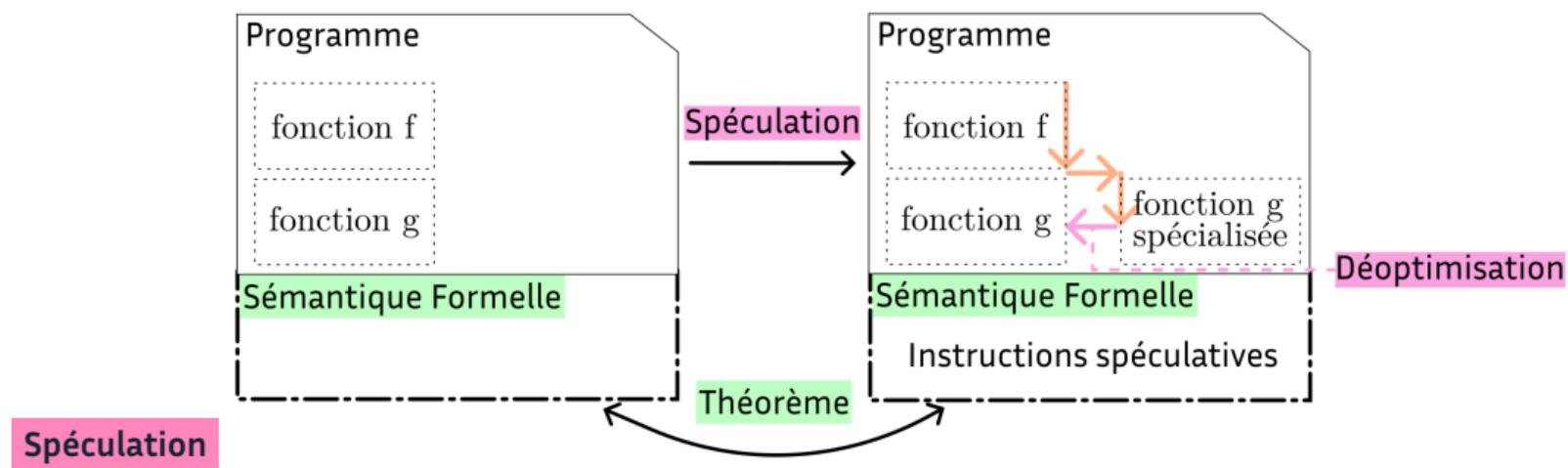
Spéculation

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.

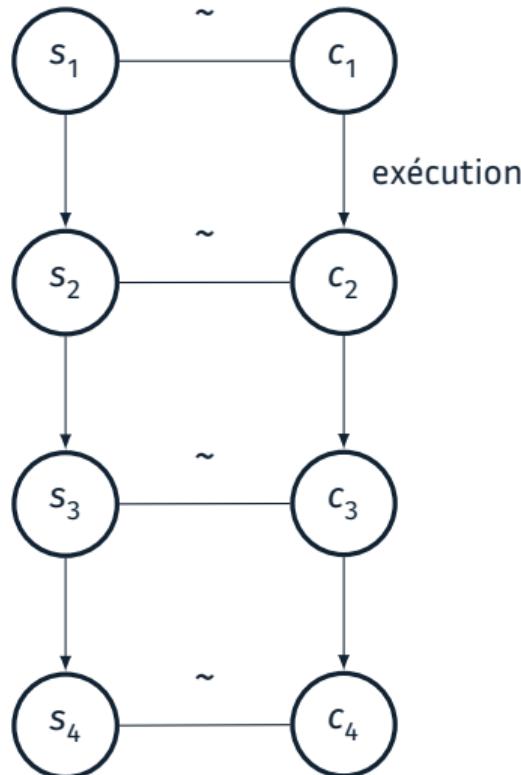
Une méthode de référence pour spéculer dans un JIT.

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

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

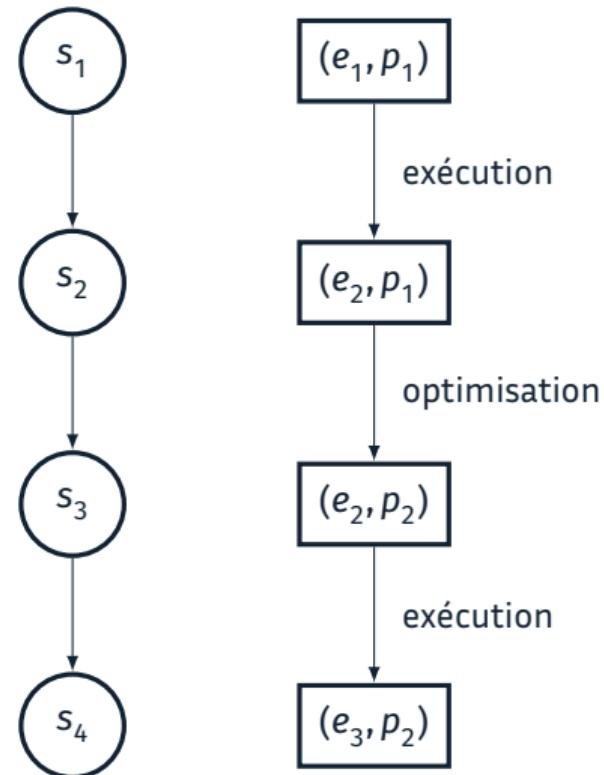
Programme SourceProgramme Compilé**Simulation :** (simplifiée)

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

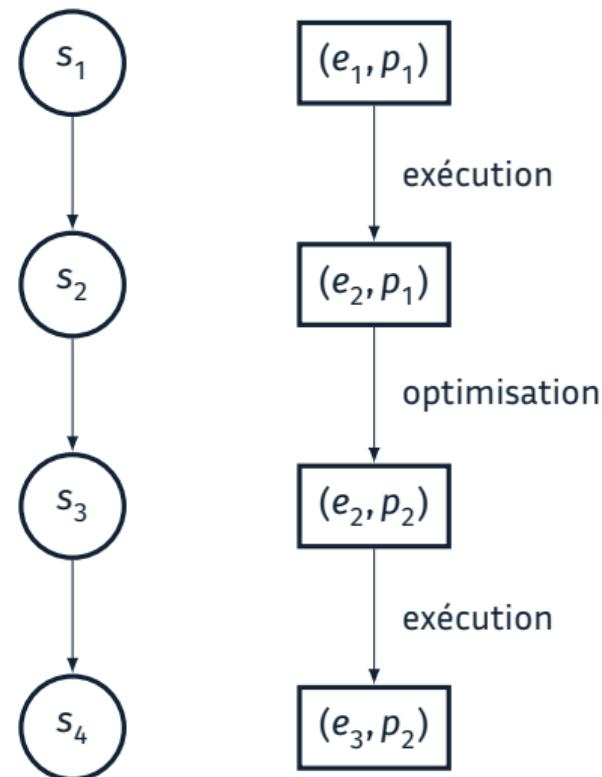


Programme SourceExécution JIT**Simulation :** (simplifiée)

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



Problème : dans un JIT, le programme change.

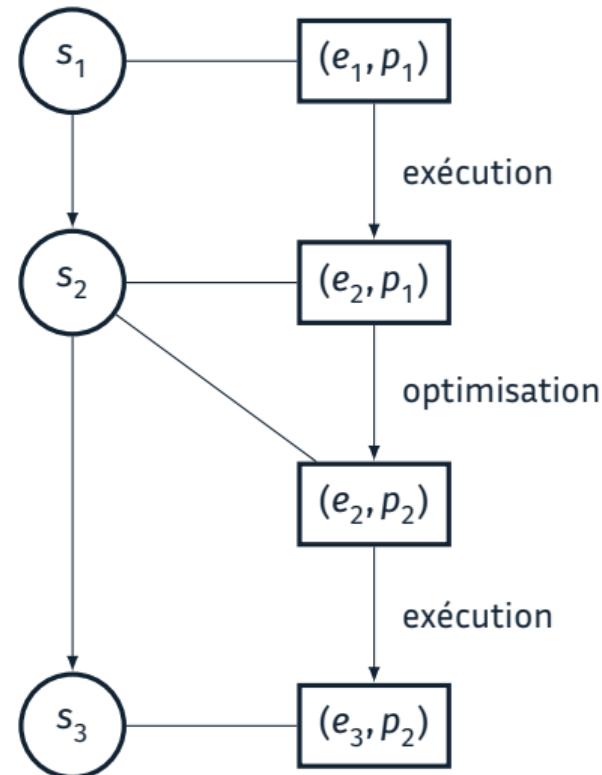
Programme SourceExécution JIT**Simulation :** (simplifiée)

- 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.

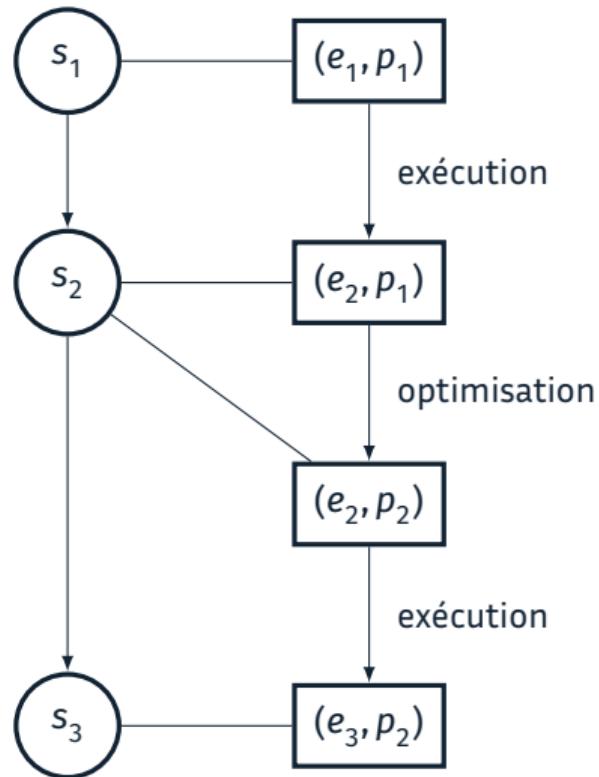
Problème : dans un JIT, le programme change.

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

interne

 $(\exists s'_1, s_1 \xrightarrow[p_1]{t} s'_1 \wedge s'_1 \sim_{int} s'_2) \text{ ou } (s_1 \sim_{int} s'_2 \wedge m_{int}(s'_2) < m_{int}(s_2) \wedge t = \emptyset)$

backward_internal_simulation $\sim_{int} m_{int} p_1 p$

externe

 $s \sim_{int} e \quad \text{backward_internal_simulation } \sim_{int} m_{int} p_1 p$

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

Mon doctorat

Des prototypes de JITs vérifiés et exécutables.

Artéfacts :

+30K lignes de Coq/Rocq

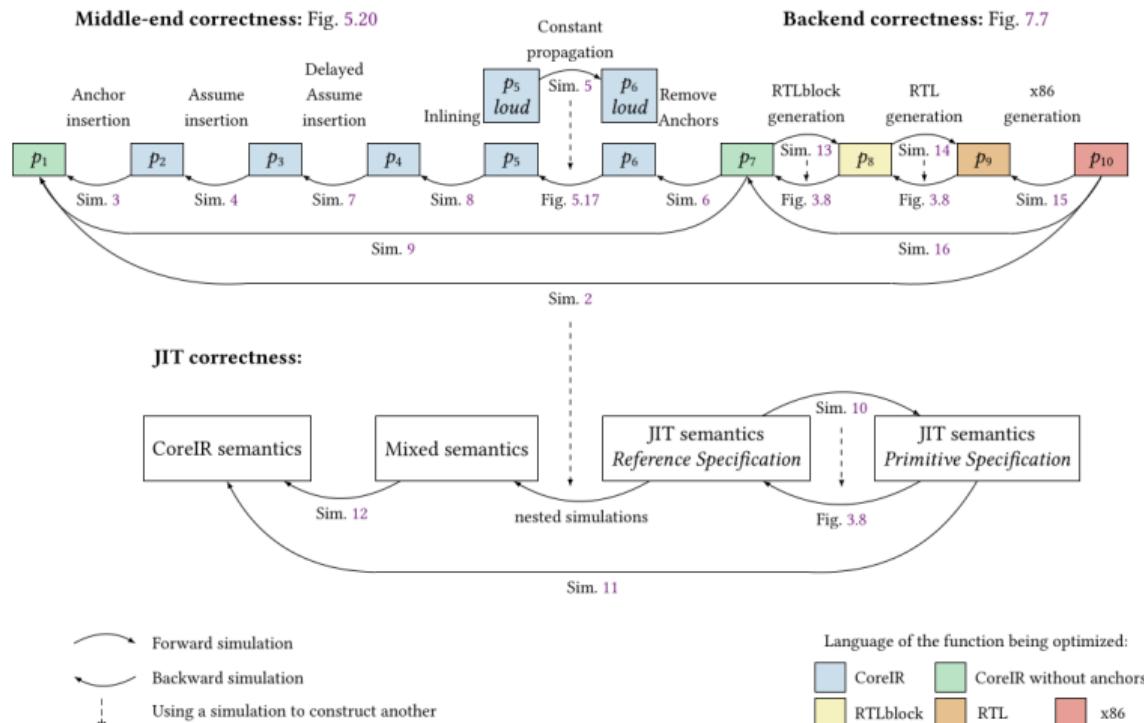


Figure 8.1 – Composing all our simulations for an effectful JIT with speculation and native code generation

$r ::=$	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/): 1014 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	$t_1 \cdot t_2$	$w = w_1 ++ w_2 \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w_2, C_{t-1}, \dots, C_{t-2}) \in \mathcal{L}_t(t_2)$
Backreference-free	t_1^*	$w = w_1 \leftrightarrow w_2 \wedge w_2 \in \mathcal{L}(t_1*) \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1)[\epsilon]$ $\wedge \{w_1 = \epsilon \wedge (w_1 = \epsilon \wedge C_1 = \dots = C_{t-1} = \emptyset)\}$
Positive Lookahead	$(?^t_1)t_2$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_{t-1}, \dots, C_{t-2}) \in \mathcal{L}_t(t_2)$
Negative Lookahead	$(!^t_1)t_2$	$(w, C_1, \dots, C_{t-1}) \notin \mathcal{L}_t(t_1) \wedge (w, C_{t-1}, \dots, C_{t-2}) \in \mathcal{L}_t(t_2)$
Input Start	t_1^+	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{\epsilon\}$
Input Start (Multiline)	t_1^*	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{\epsilon\}$
Input End	$\$t_1$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t \wedge \{\epsilon\}$
Input End (Multiline)	$\$t_1$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t[\epsilon]\wedge \{\epsilon\}$ $w = w_1 ++ w_2 \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w_2, C_{t-1}, \dots, C_{t-2}) \in \mathcal{L}_t(t_2)$ $\wedge \{(\{w_1 \in \mathcal{L}[\epsilon] \wedge w_1 = \epsilon\} \vee w_2 \in \mathcal{L}[\epsilon]) \wedge (\{w_1 \in \mathcal{L}[\epsilon] \wedge (w_2 \in \mathcal{L}[\epsilon] \wedge w_2 = \epsilon)\})\}$
Word Boundary	$t_1(B)t_2$	$w = w_1 \wedge (w_1 \in \mathcal{L}[\epsilon] \wedge w_1 = \epsilon) \wedge w_2 \in \mathcal{L}[\epsilon] \wedge \{w_1 \in \mathcal{L}[\epsilon] \wedge (w_2 \in \mathcal{L}[\epsilon] \wedge w_2 = \epsilon)\}$ $\wedge \{(\{w_1 \notin \mathcal{L}[\epsilon] \wedge w_1 \neq \epsilon\} \wedge w_2 \notin \mathcal{L}[\epsilon]) \wedge (\{w_1 \notin \mathcal{L}[\epsilon] \wedge w_1 \neq \epsilon\} \wedge \{w_2 \notin \mathcal{L}[\epsilon] \wedge w_2 \neq \epsilon\})\}$
Non-Word Boundary	$t_1(B)t_2$	$w = w_1 \wedge (w_1 \in \mathcal{L}[\epsilon] \wedge w_1 = \epsilon) \wedge w_2 \in \mathcal{L}[\epsilon] \wedge \{w_1 \in \mathcal{L}[\epsilon] \wedge (w_2 \in \mathcal{L}[\epsilon] \wedge w_2 = \epsilon)\}$ $\wedge \{(\{w_1 \notin \mathcal{L}[\epsilon] \wedge w_1 \neq \epsilon\} \wedge w_2 \notin \mathcal{L}[\epsilon]) \wedge (\{w_1 \notin \mathcal{L}[\epsilon] \wedge w_1 \neq \epsilon\} \wedge \{w_2 \notin \mathcal{L}[\epsilon] \wedge w_2 \neq \epsilon\})\}$
Capture Group	$(?_1)$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge C_0 = w$
Non-Capturing Group	$(?:_1)$	$(w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1)$
Base Case	$t \text{ regular}$	$w \in \mathcal{Q}(t)$

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	$t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge C_{t-1} = \dots = C_{t-1} = \emptyset \rangle$ $\vee \langle (w, C_{t+1}, \dots, C_{t+1}) \in \mathcal{L}_t(t_2) \wedge C_t = \dots = C_{t-1} = \emptyset \rangle$
Concaténation	$t_1 \cdot t_2$	$w = w_1 \cdots w_t \wedge w_i \in \mathcal{L}(t_i) \wedge C_t = \dots = C_{t-1} = \emptyset$
Backreference-free	t_1^*	$w = w_1 \cdots w_t \wedge w_i \in \mathcal{L}(t_1) \wedge (w_1, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1)$ $\wedge (w_t = \epsilon \iff (w_1 = \epsilon \wedge \dots \wedge C_{t-1} = \emptyset))$
Quantification	$(?^n)t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_2) \rangle$
Positive Lookahead	$(?^n)t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \notin \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_2) \rangle$
Negative Lookahead	$(?^n)t_1 t_2$	$\langle (w, C_1, \dots, C_{t-1}) \notin \mathcal{L}_t(t_1) \wedge (w, C_1, \dots, C_{t-1}) \in \mathcal{L}_t(t_2) \rangle$
Empty Set	\emptyset	$w[p] = a$
Character	$(a, w, p, \Lambda) \rightsquigarrow \langle (p+1, \Lambda) \rangle$	$(r, w, p, \Lambda) \rightsquigarrow \mathcal{N}$
Character Failure	$(a, w, p, \Lambda) \rightsquigarrow \emptyset$	$\langle (r, j), w, p, \Lambda \rightsquigarrow \{(p_k, \Lambda_k j \rightsquigarrow w[p-p_k]) \mid (p_k, \Lambda_k) \in \mathcal{N}\} \rangle$
Capturing Group	$(\bar{A}, w, p, \Lambda) \rightsquigarrow \emptyset$	$(\bar{A}, w, p, \Lambda) \rightsquigarrow \emptyset$
Empty String	$(\emptyset, w, p, \Lambda) \rightsquigarrow \{(p, \Lambda)\}$	$\langle (\lambda(i), w, p, \Lambda) \rightsquigarrow \mathcal{N} \rangle$
Backreference	$(r_1, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \quad \forall (p_k, \Lambda_k) \in \mathcal{N}, (r_2, w, p, \Lambda_k) \in \mathcal{N}'$	$\langle (\lambda(j), w, p, \Lambda) \rightsquigarrow \mathcal{N}' \rangle$
Backreference Failure	$(r_1r_2, w, p, \Lambda) \rightsquigarrow \bigcup_{1 \leq i < j \leq \mathcal{N} } \mathcal{N}_i$	$\langle (\lambda(w, p, \Lambda) \rightsquigarrow \emptyset) \rangle$
Concatenation	$(r_1, w, p, \Lambda) \rightsquigarrow \mathcal{N}'$	$(r, w, p, \Lambda) \rightsquigarrow \mathcal{N}$
Positive Lookahead	$(r_1r_2, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \cup \mathcal{N}''$	$\langle (\lambda(r), w, p, \Lambda) \rightsquigarrow \{(p, \Lambda' \dots, \Lambda') \in \mathcal{N}\} \rangle$
Union	$(r_1, w, p, \Lambda) \rightsquigarrow \mathcal{N}'$	$\langle (r, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \rangle$
Negative Lookahead	$(r_1r_2, w, p, \Lambda) \rightsquigarrow \mathcal{N}' \cup \mathcal{N}''$	$\langle (\lambda(\neg r), w, p, \Lambda) \rightsquigarrow \mathcal{N}'' \rangle$
Repetition	$\forall (p_k, \Lambda_k) \in (\mathcal{N}' \setminus \{(p, \Lambda)\}), (r^*, w, p, \Lambda_k) \rightsquigarrow \mathcal{N}'$	$\langle (r^*, w, p, \Lambda) \rightsquigarrow \{(p, \Lambda) \cup \bigcup_{1 \leq i < \mathcal{N}' \setminus \{(p, \Lambda)\}} \mathcal{N}_i \} \rangle$

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 3: 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 () 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 *s* be the *MatchState* ([|*Input*|]; *Input*, [|EndIndex|]; *index*, [|Captures|]; *cap*).
f. Return *m(x, c)*.

Non équivalent

Modèles existants

Operation	t	Overapproximate Model for $(w, C_1, \dots, C_k) \in \mathcal{L}_t(t)$
Alternation	$t_1 t_2$	$\{w, (C_1, \dots, C_k) \in \mathcal{L}_1(t) \mid (C_1, \dots, C_k) = \emptyset\} \cup \{w, (C_1, \dots, C_k) \in \mathcal{L}_2(t) \mid (C_1, \dots, C_k) = \emptyset\}$
Concatenation	$t_1 \cdot t_2$	$w = w_1 \cdots w_n \wedge (w_1, C_1, \dots, C_k) \in \mathcal{L}_1(t) \wedge \dots \wedge (w_n, C_1, \dots, C_k) \in \mathcal{L}_1(t)$
Backreference-free Quantification	$t_{\forall x}$	$w = w_1 \cdots w_n \wedge \forall x \in \mathcal{L}(t) \forall x \wedge (w_1, C_1, \dots, C_k) \in \mathcal{L}_1(t_1)$ $\wedge \exists y \in \mathcal{L}(t) \forall x \wedge (w_1, C_1, \dots, C_k) \in \mathcal{L}_1(t_1)$
Positive Lookahead	$(\exists t_1) t_2$	$(w, C_1, \dots, C_k) \in \mathcal{L}_1(t_1) \wedge (w, C_1, \dots, C_k) \in \mathcal{L}_2(t_2)$
Negative Lookahead	$(\nexists t_1) t_2$	$(w, C_1, \dots, C_k) \notin \mathcal{L}_1(t_1) \wedge (w, C_1, \dots, C_k) \in \mathcal{L}_2(t_2)$
Quantifier Negation	$\neg t$	$\neg(w, C_1, \dots, C_k) \in \mathcal{L}_t(t)$
1	$p \leq w $	$w[p] = a$
1	$(a, w, p, \lambda) \rightarrow$	(CHARACTER)
1	$p \geq v \wedge v[p] \neq a$	$(r, w, p, \lambda) \rightarrow N$
1	$(a, w, p, \lambda) \rightarrow \emptyset$	(CHARACTER FAILURE)
1		(CAPTURING GS)
1	$(\emptyset, w, p, \lambda) \rightarrow \emptyset$	(EMPTY SET)
2	$(a_1, w, p, \lambda) \rightarrow [p, a_1]$	(EMPTY STRING)
2	$(r_1, w, p, \lambda) \rightarrow N$	$(r, w, p, \lambda) \rightarrow N$
2	$(r_1 r_2, w, p, \lambda) \rightarrow \bigcup_{1 \leq i \leq j \leq 2} N_i$	(BACKREFERENCE FAIL)
2	$(r_1 r_2, w, p, \lambda) \rightarrow \emptyset$	($(\emptyset, w, p, \lambda) \rightarrow \emptyset$)
2		(CAPTIONING GS)
2	$(r_1, w, p, \lambda) \rightarrow N'$	(CONCATENATION)
2	$(r_1 r_2, w, p, \lambda) \rightarrow N' \cup N'_2$	($(N', w, p, \lambda) \rightarrow [p, N']$)
2	$(r_1, w, p, \lambda) \rightarrow N'$	(UNION)
2	$(r_1 r_2, w, p, \lambda) \rightarrow N' \cup N'_2$	($(N', w, p, \lambda) \rightarrow N'$)
2	$\forall (p_1, \lambda) \in \{N'_1(N'), N'_2(N')\}$	(POSITIVE LOOKAHEAD)
2	$(r^*, w, p, \lambda) \rightarrow \{p, \lambda\} \cup \bigcup_{1 \leq i \leq k} N_i$	($(\{p, \lambda\}, w, p, \lambda) \rightarrow N$)
2		(REPETITION)
2	$\forall (p_1, \lambda) \in \{N'_1(N'), N'_2(N')\}$	(NEGATIVE LOOKAHEAD)

■ 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 :: *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 :



Nouvelle sémantique

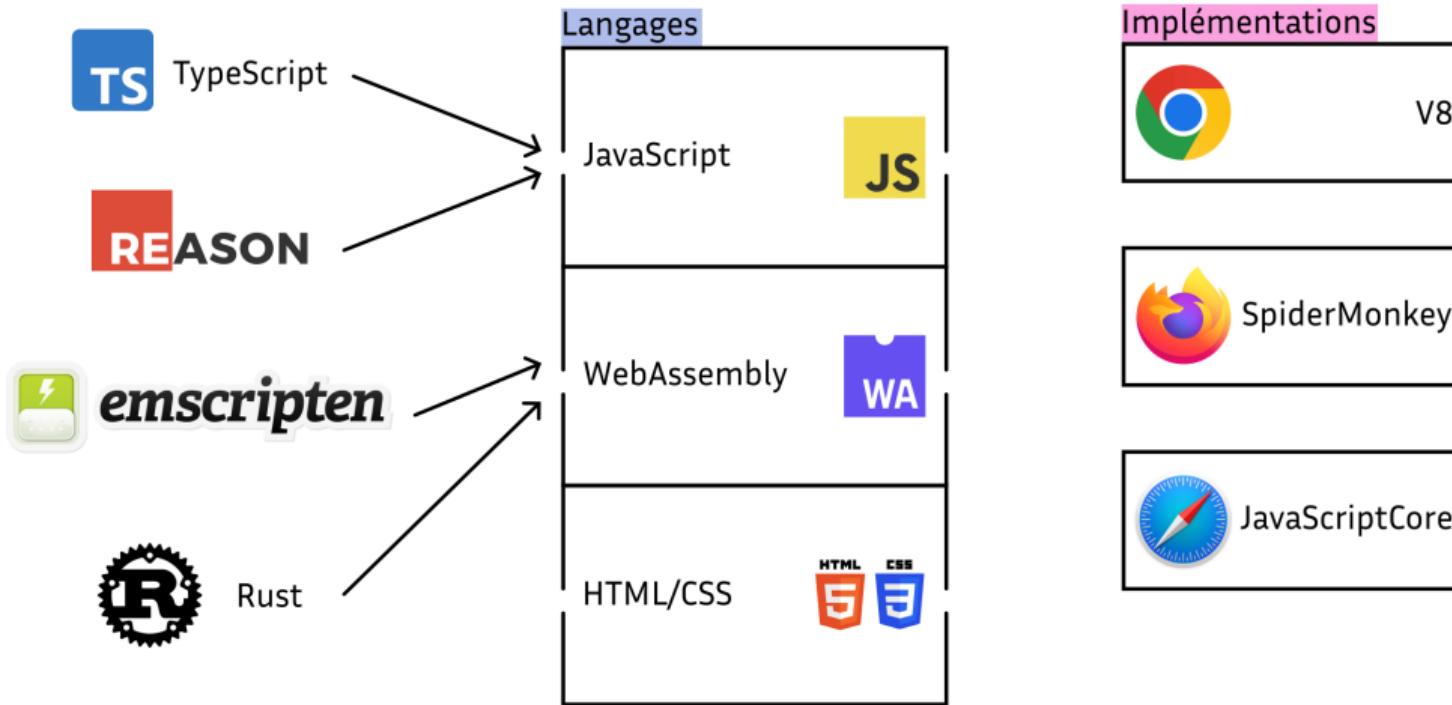
On peut enfin vérifier formellement des moteurs de regex JavaScript !

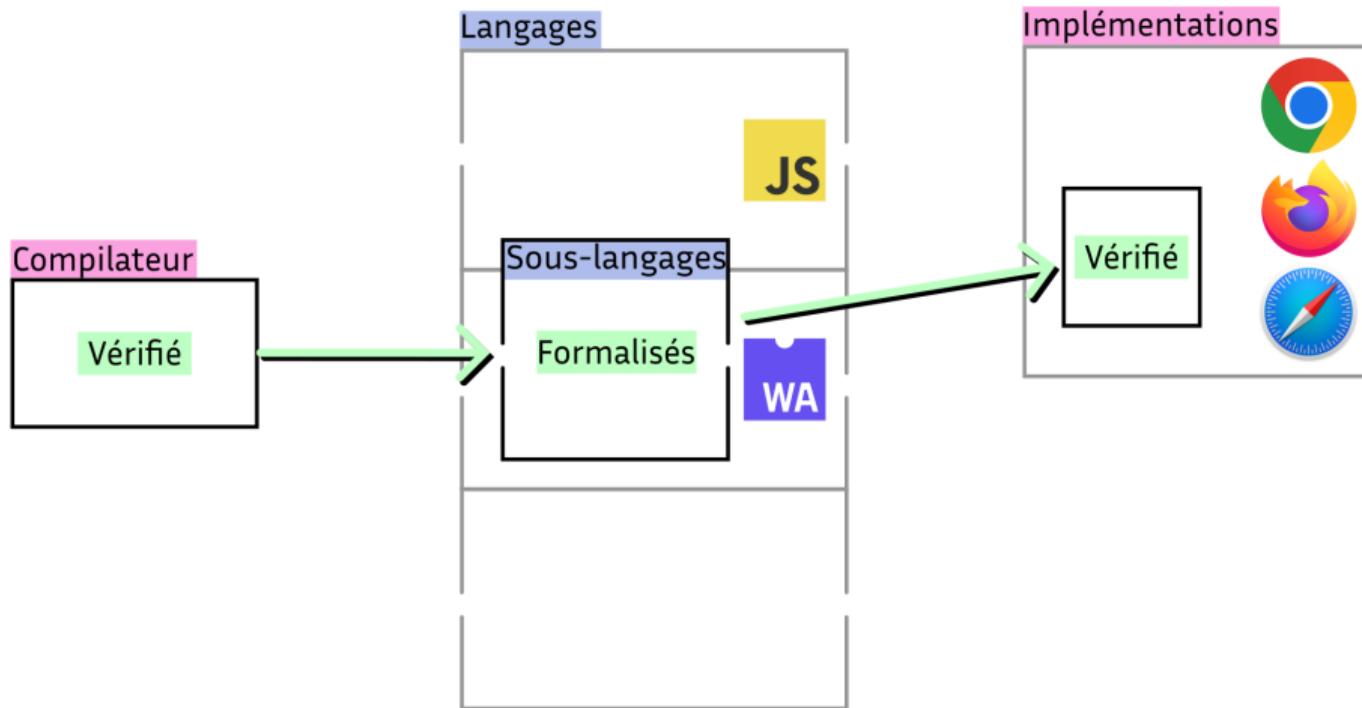
Publications : [PLDI'24], [ICFP'24]
Encadrement de 12 projets d'étudiants.

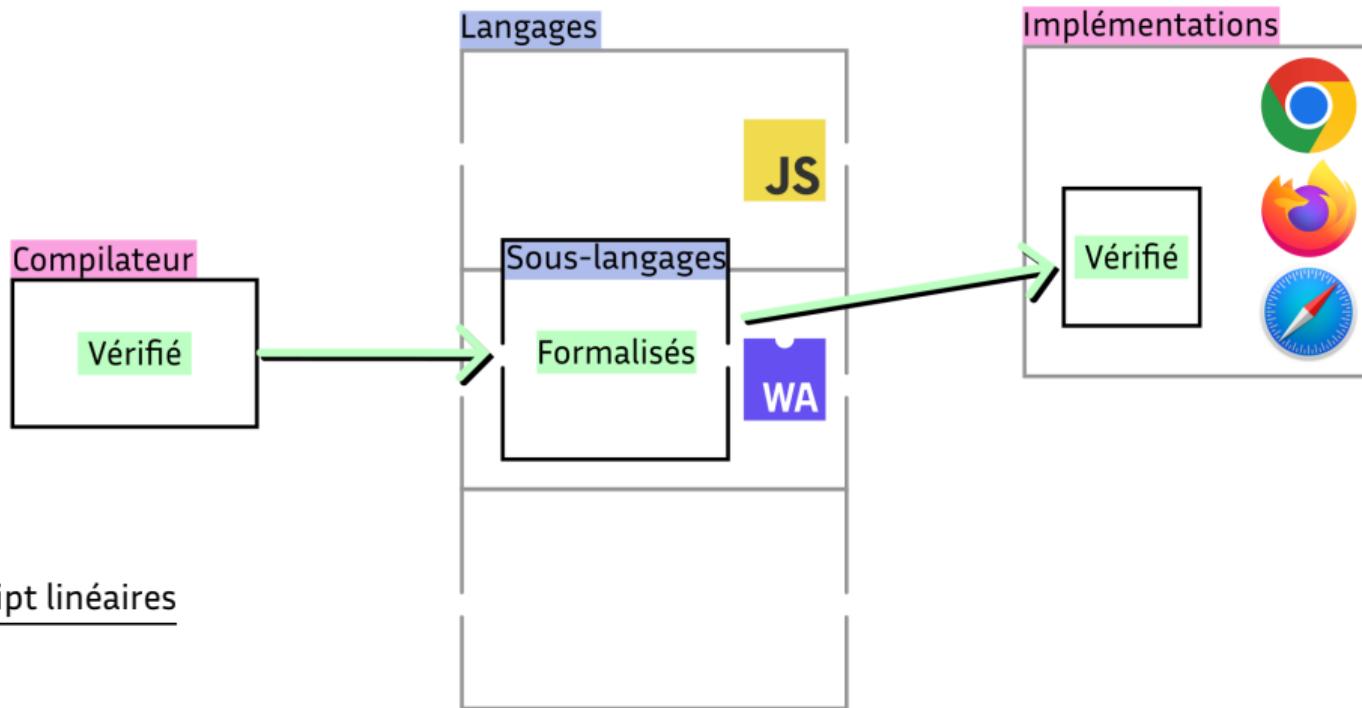
Financements :
Open Research Data Contribute Grant
Swiss National Science Foundation Project

Programme de Recherche

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



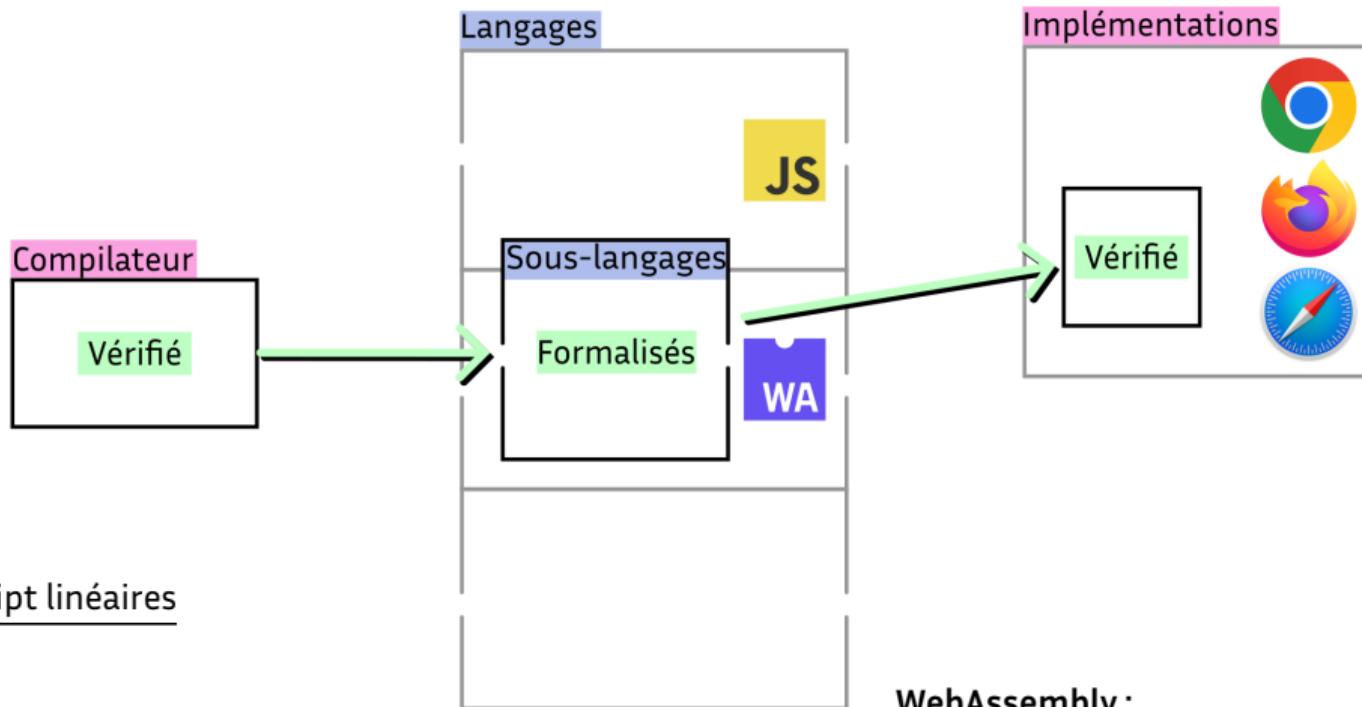




Deux axes :

Regex JavaScript linéaires

Un sous-ensemble de WebAssembly



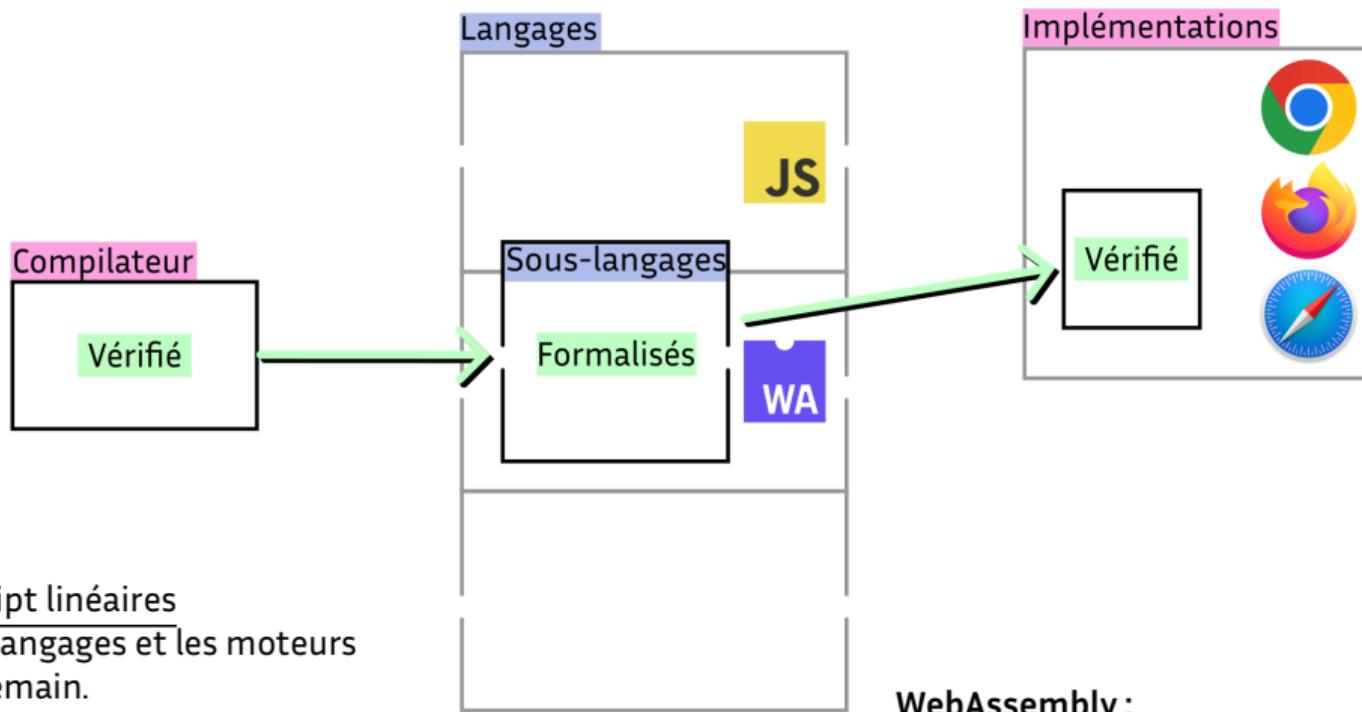
Deux axes :

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

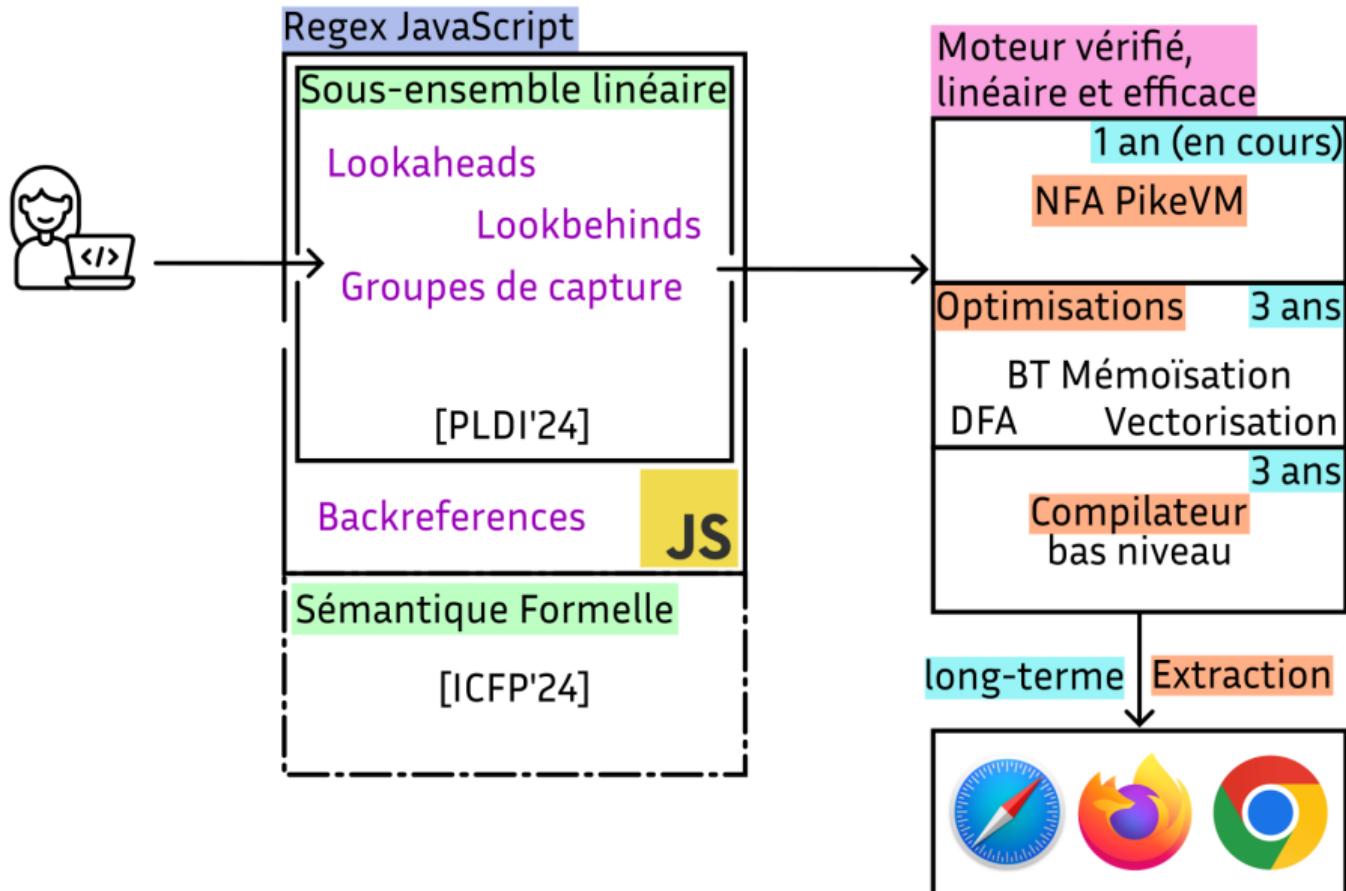
[PLDI'24]

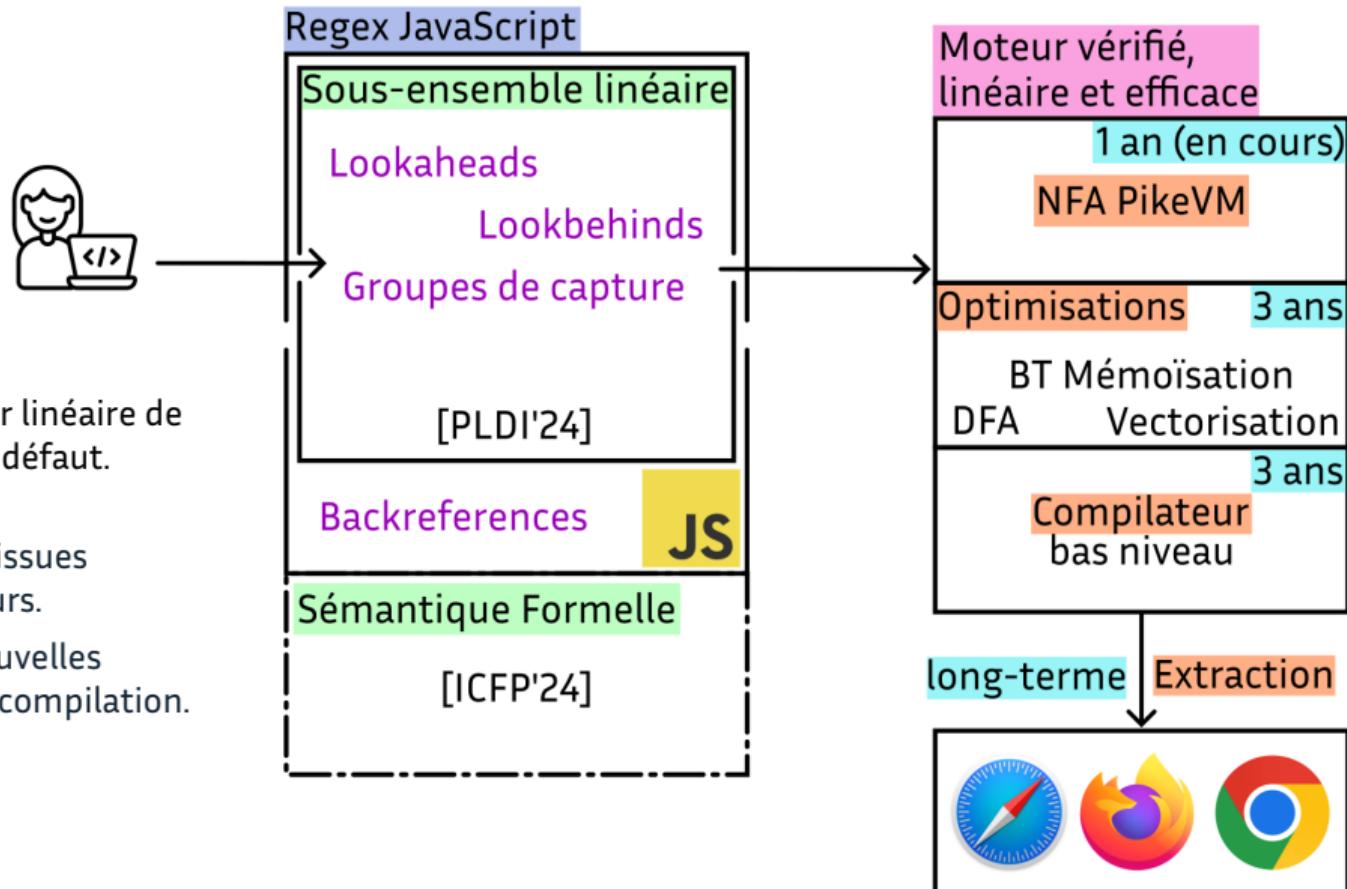
Backreferences

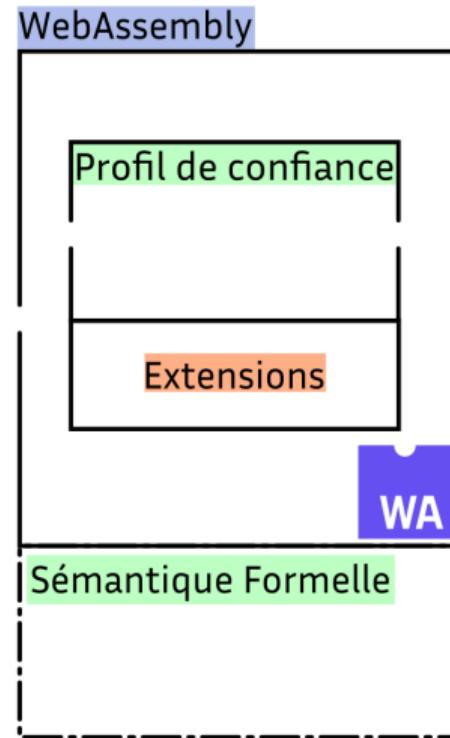
JS

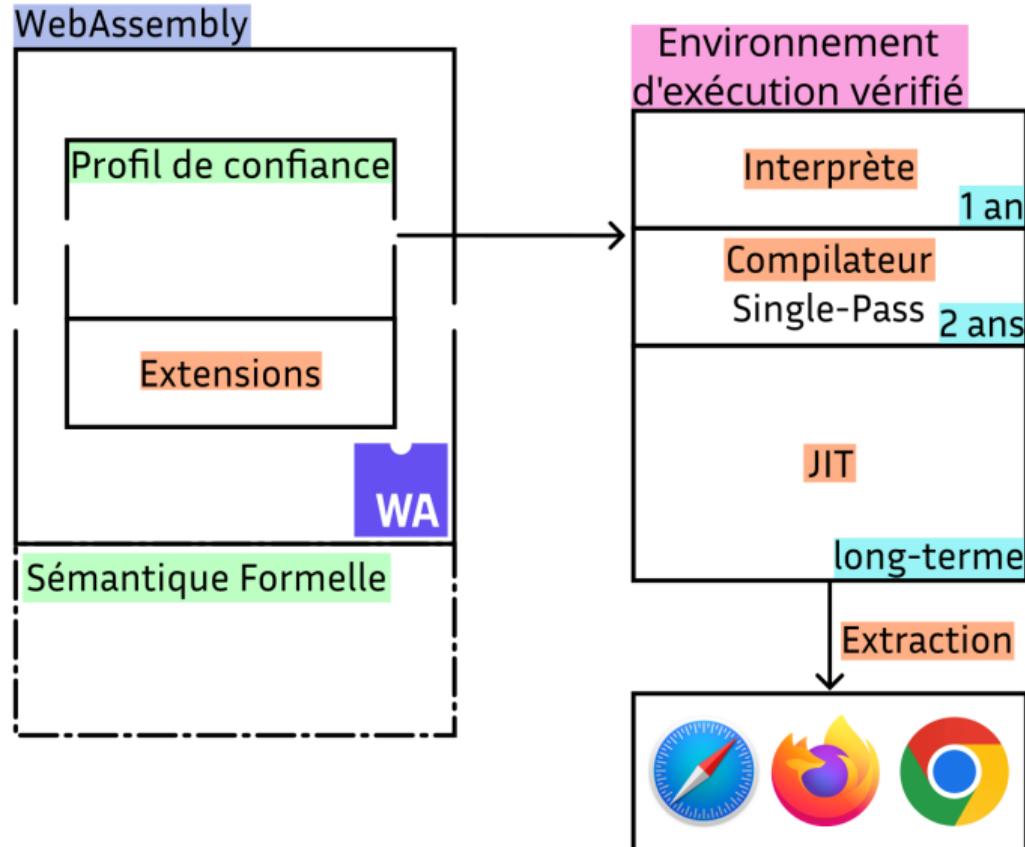
Sémantique Formelle

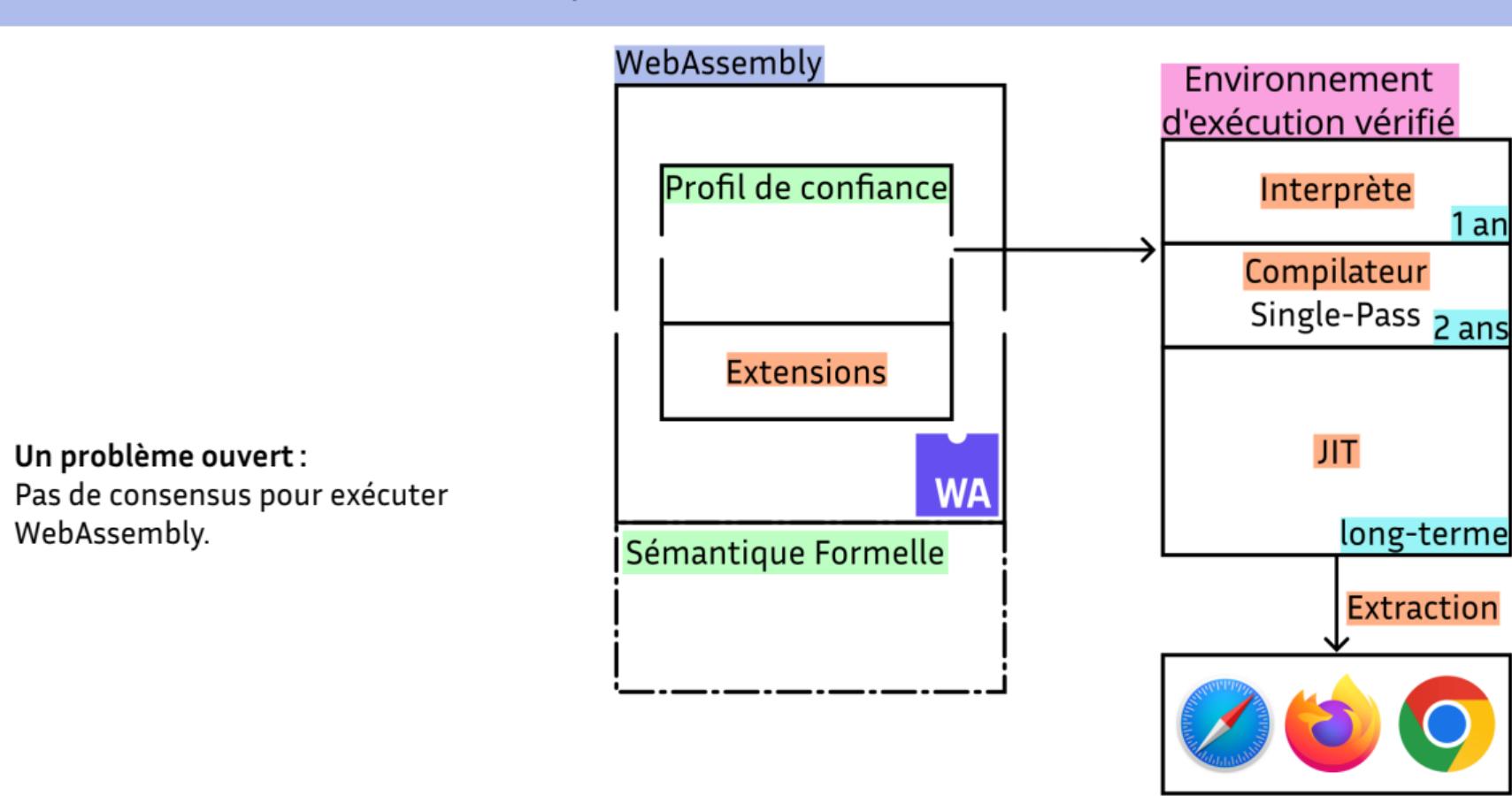
[ICFP'24]

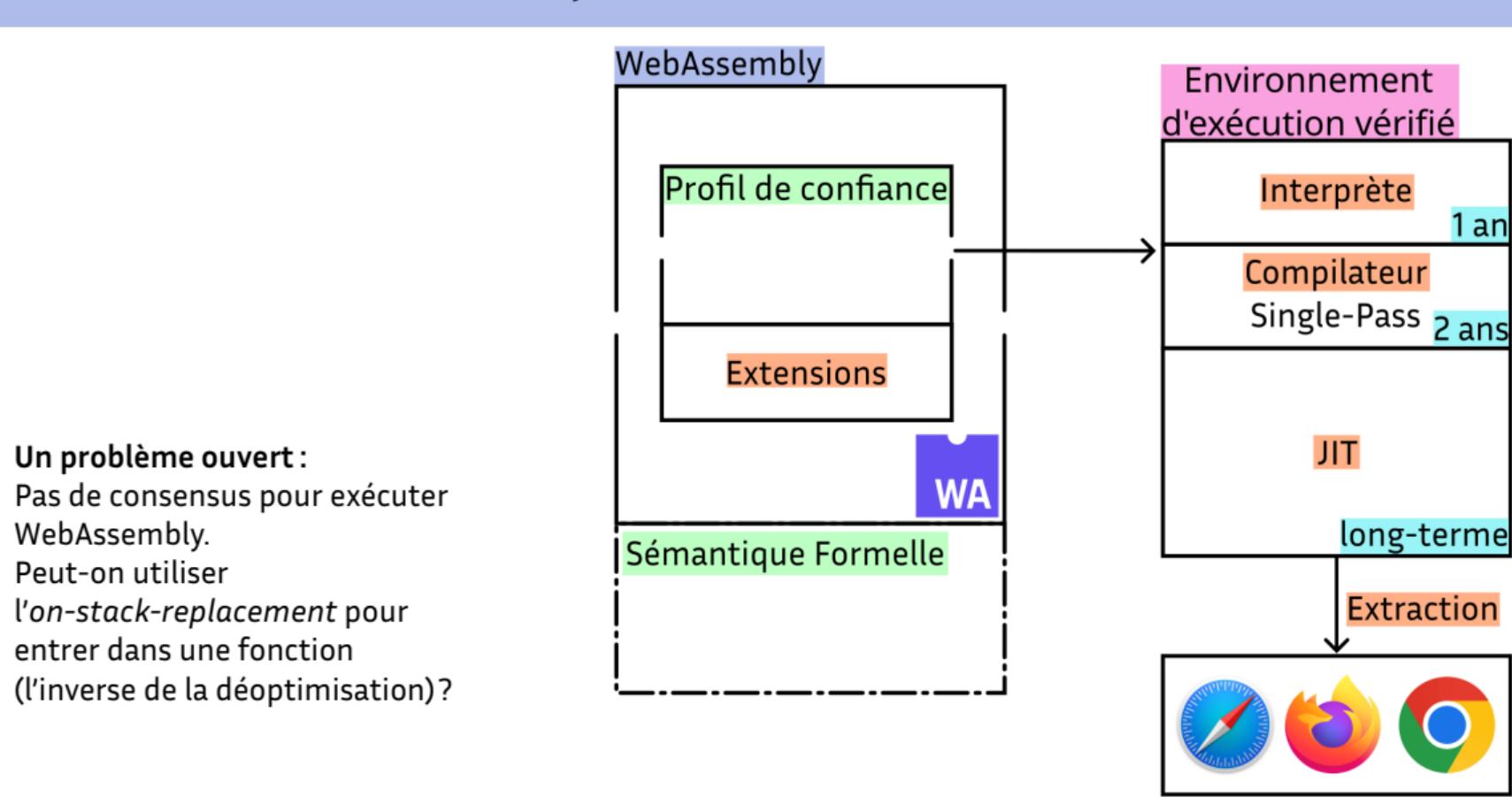


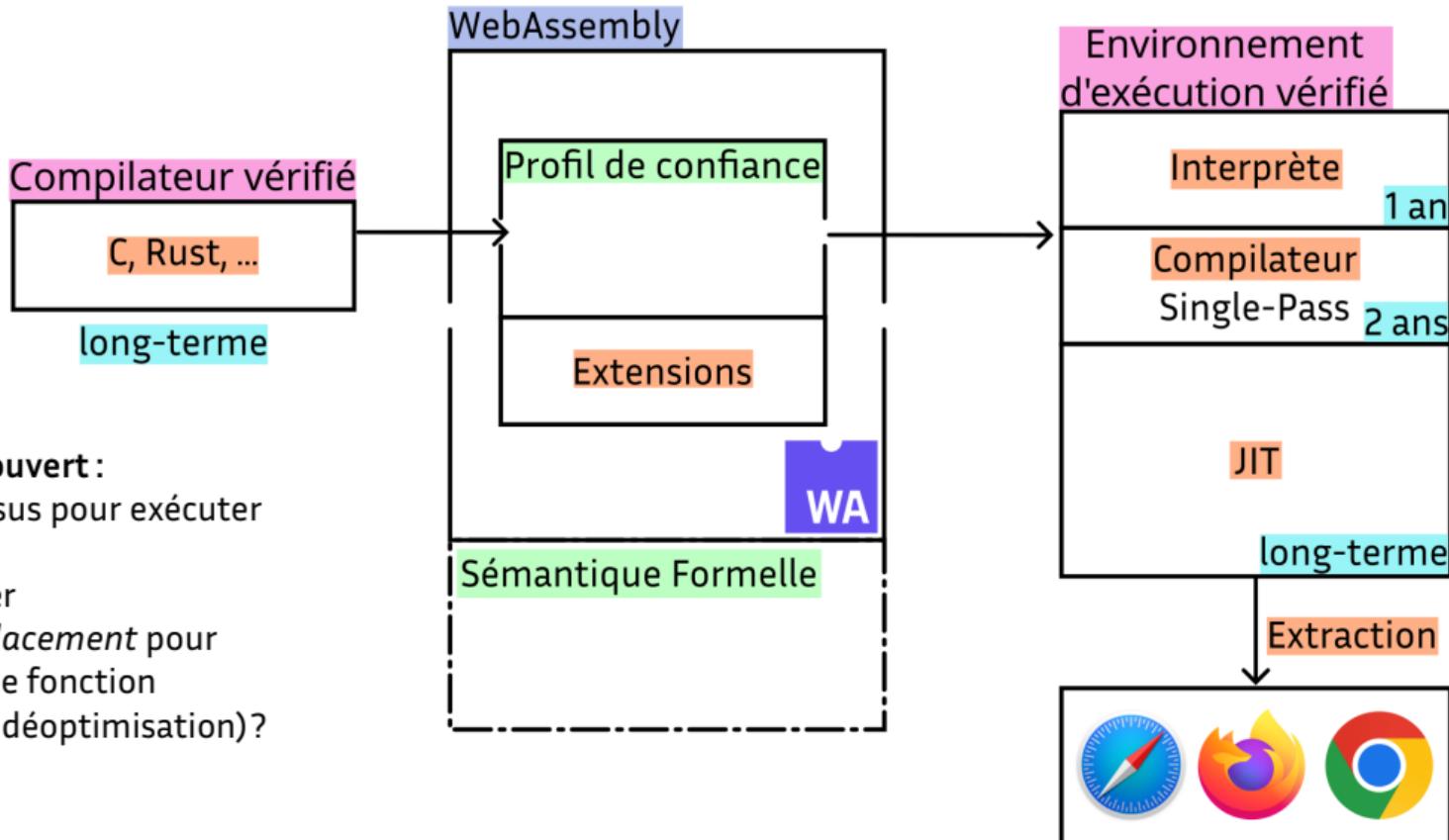


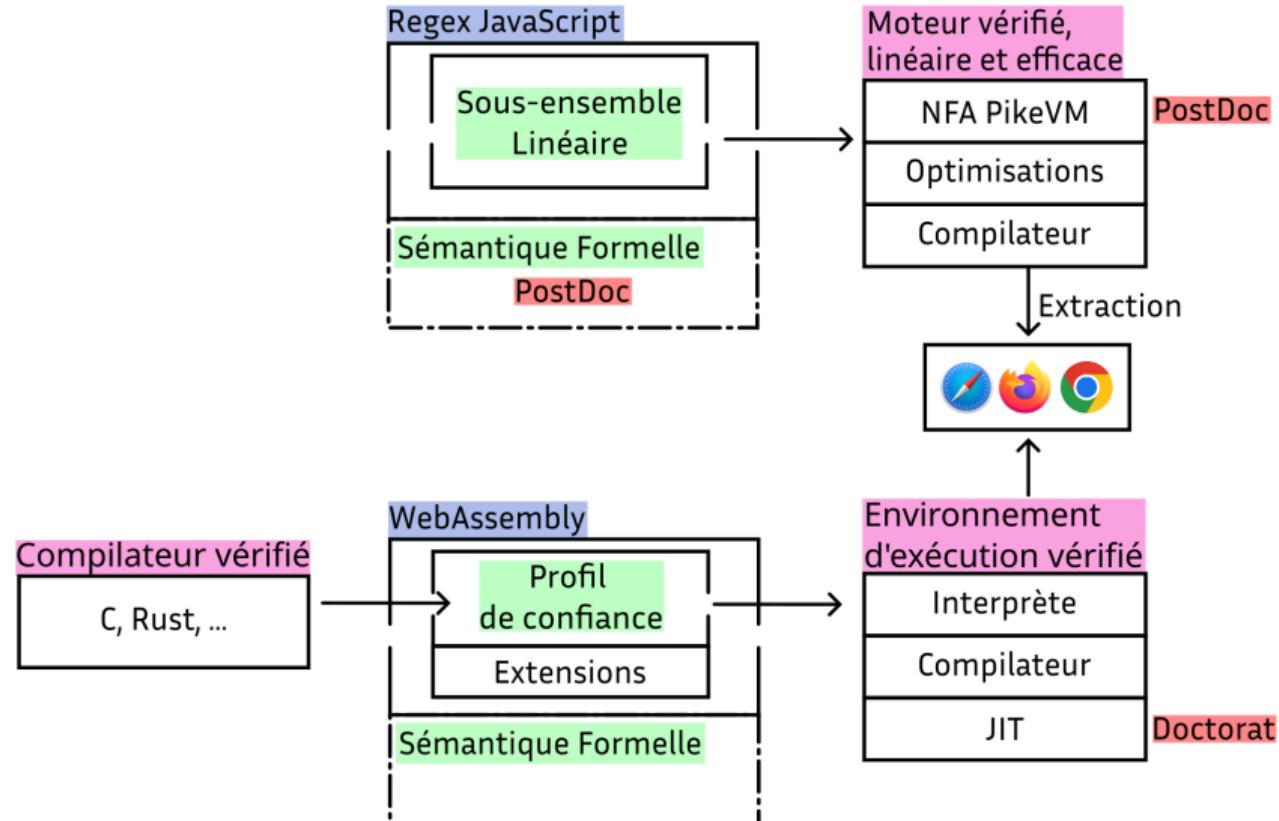












Cristal

Vectorisation formalisée (C. Paperman)

Extraction vérifiée (V. Rusu, D. Nowak)

Sémantique multi-langages (R. Monat)

LIP

Vectorisation formalisée (G. Radanne)

Compilation vérifiée (Y. Zakowski)

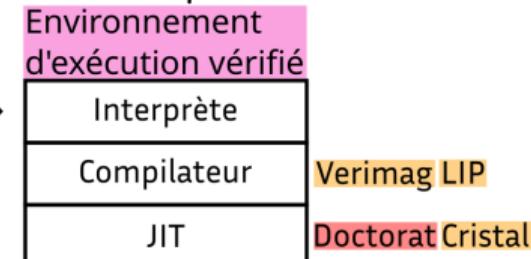
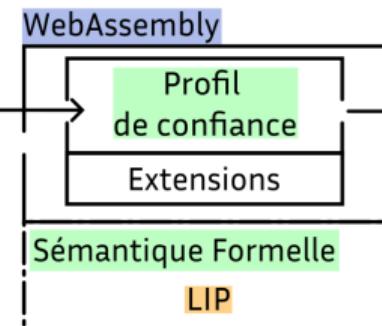
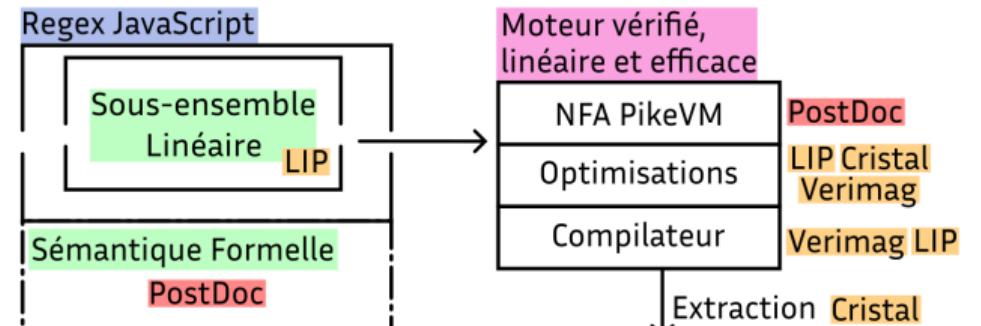
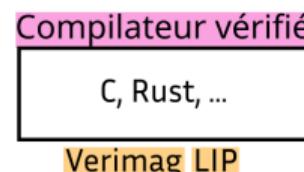
Sémantique concurrente (L. Henrio)

Verimag

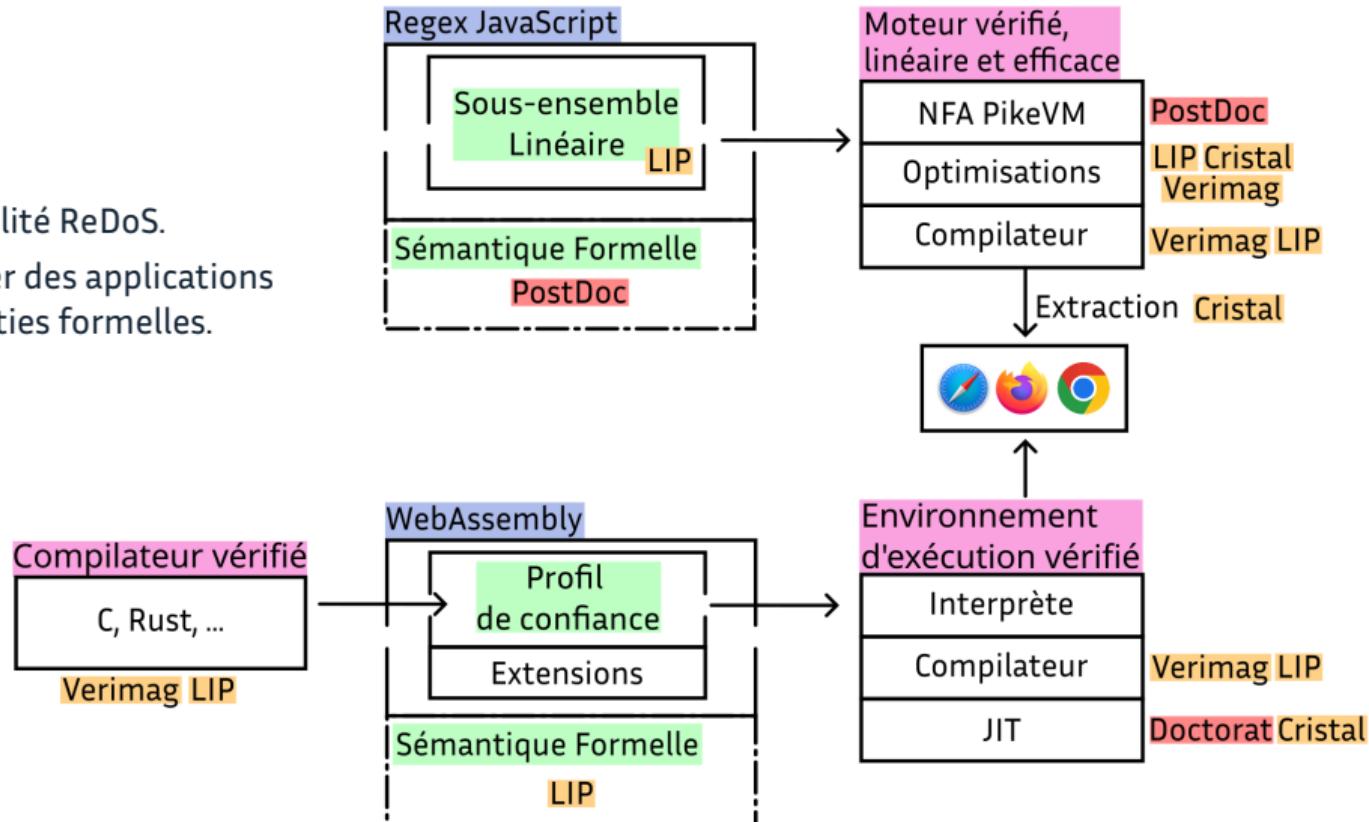
Vectorisation formalisée (B. Ferres)

Compilation vérifiée (D. Monniaux, S. Boulmé)

Compilation de Rust (D. Monniaux, S. Boulmé)



- Vaincre la vulnérabilité ReDoS.
- Déployer et exécuter des applications web avec des garanties formelles.



Transparents supplémentaires

1/16 Mon domaine

2/16 Compilation formellement vérifiée

3/16 Écosystème web, compilation non traditionnelle

4/16 Ma méthodologie

5/16 Doctorat

6/16 Spéculation & Déoptimisation

7/16 Simulations imbriquées

8/16 JITs vérifiés

9/16 PostDoc

10/16 Nouveaux algorithmes linéaires

11/16 Nouvelle sémantique des regex

12/16 Avancements algorithmiques et sémantiques

13/16 Vers une plateforme web de confiance

14/16 Un moteur vérifié, linéaire, efficace, intégrable

15/16 Un sous-ensemble WebAssembly de confiance

16/16 Vérification formelle pour un Web de confiance

Transparents supplémentaires :

Spéculer dans un langage dynamique

Insérer des instructions spéculatives

Définition Simulations Imbriquées

Simulations imbriquées, exécution

Simulations imbriquées, optimisation

Regex modernes avec priorité

L'étoile JavaScript est unique

Simulation de NFA et étoile

Dupliquer le graphe pour l'étoile JavaScript

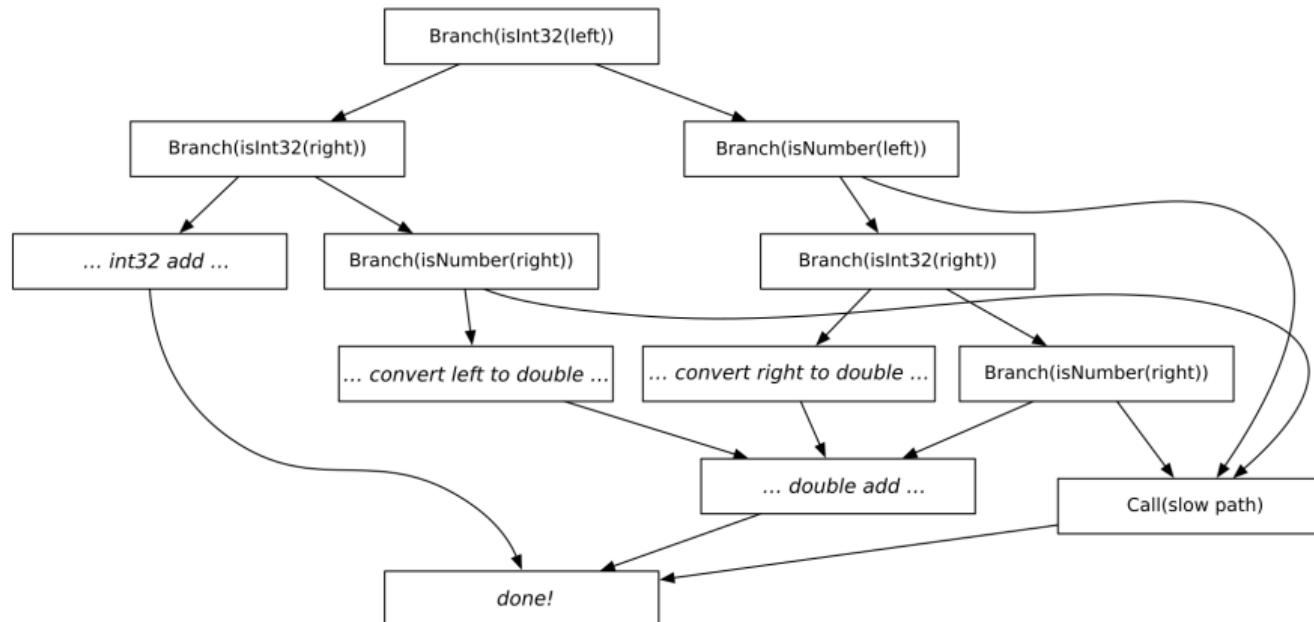
Lookarounds et groupes de capture

3 étapes pour les lookarounds

Une mécanisation de confiance

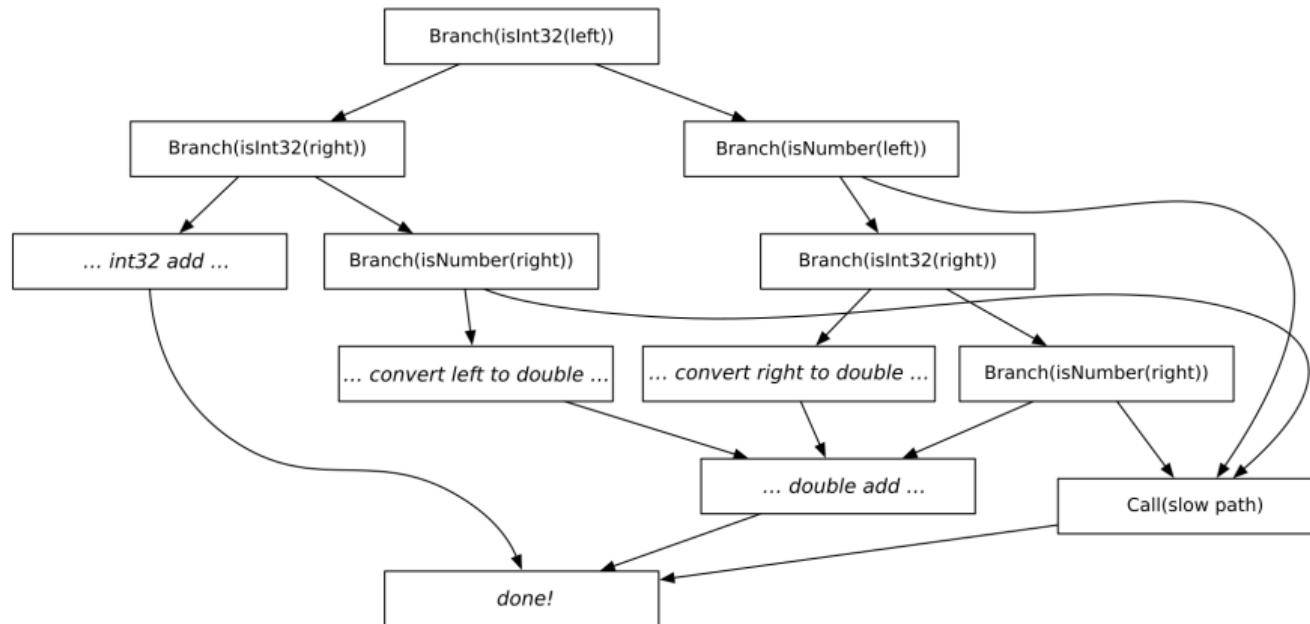
Pour exécuter `left + right`, un exemple issu de JavaScriptCore.

Pour exécuter `left + right`, un exemple issu de JavaScriptCore.



Difficile à compiler.
Restreint des optimisations.

Pour exécuter `left + right`, un exemple issu de JavaScriptCore.



Difficile à compiler.
Restreint des optimisations.

Spéculation de type

En spéculant sur les types de `left` et `right`, le graphe est réduit à un seul nœud. Un seul test si les variables sont utilisées plusieurs fois.

```
Function F (x, y):
```

```
Version Base:
```

```
l1: a ← 1
```

```
b ← y + x
```

```
Return (a + b)
```

```
Function F (x, y):  
Version Base:  
l1: a ← 1
```

b ← y + x
Return (a + b)

JIT : il est possible qu'on spécle ici plus tard

Function F (x, y):
Version Base:
l1: a ← 1

Version Opt:
l1: a ← 1
Anchor F.l1 [x ← x, y ← y]

b ← y + x
b ← y + x

Return (a + b)

Return (a + b)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor

JIT : il est possible qu'on spécle ici plus tard

Function F (x, y):
Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

l1: a ← 1

Anchor F.l1 [x ← x, y ← y]

b ← y + x

Return (a + b)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor

JIT : spéculons que x est égal à 9

Function F (x, y):

Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

l1: a ← 1

Anchor F.l1 [x ← x, y ← y]

Assume (x=9) F.l1 [x ← x, y ← y]

b ← y + x

Return (a + b)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor
- Insérer spéculation, Assume

JIT : spéculons que x est égal à 9

Function F (x, y):

Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

l1: a ← 1

Anchor F.l1 [x ← x, y ← y]

Assume (x=9) F.l1 [x ← x, y ← y]

b ← y + 9

Return (1 + b)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor
- Insérer spéculation, Assume
- Propagation de constantes

Function F (x, y):

Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

l1: a ← 1

Anchor F.l1 [x ← x, y ← y]

Assume (x=9) F.l1 [x ← x, y ← y]

Assume (y=7) F.l1 [x ← x, y ← y]

b ← y + 9

Return (1 + b)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor
- Insérer spéculation, Assume
- Propagation de constantes
- Insérer spéculation

JIT : spéculons que y est égal à 7

Function F (x, y):

Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

l1: a ← 1

Anchor F.l1 [x ← x, y ← y]

Assume (x=9) F.l1 [x ← x, y ← y]

Assume (y=7) F.l1 [x ← x, y ← y]

b ← 16

Return (17)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor
- Insérer spéculation, Assume
- Propagation de constantes
- Insérer spéculation
- Propagation de constantes

Function F (x, y):

Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

Anchor F.l1 [x ← x, y ← y]
Assume (x=9) F.l1 [x ← x, y ← y]
Assume (y=7) F.l1 [x ← x, y ← y]

Return (17)

Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor
- Insérer spéculation, Assume
- Propagation de constantes
- Insérer spéculation
- Propagation de constantes
- Élimination de code mort

Function F (x, y):

Version Base:

l1: a ← 1

b ← y + x

Return (a + b)

Version Opt:

Anchor F.l1 [x ← x, y ← y]
Assume (x=9) F.l1 [x ← x, y ← y]
Assume (y=7) F.l1 [x ← x, y ← y]

Return (17)

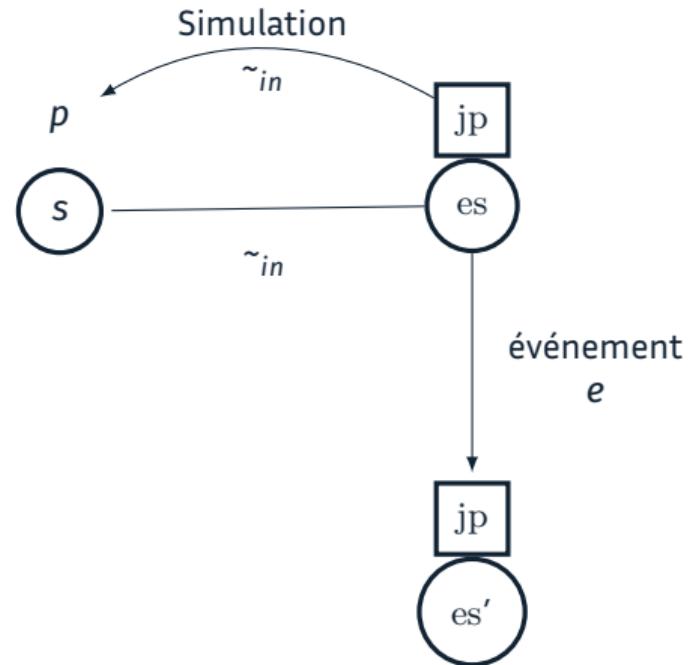
Timeline :

- Créer une version Opt, avec des points de synchronisation, Anchor
- Insérer spéculation, Assume
- Propagation de constantes
- Insérer spéculation
- Propagation de constantes
- Élimination de code mort

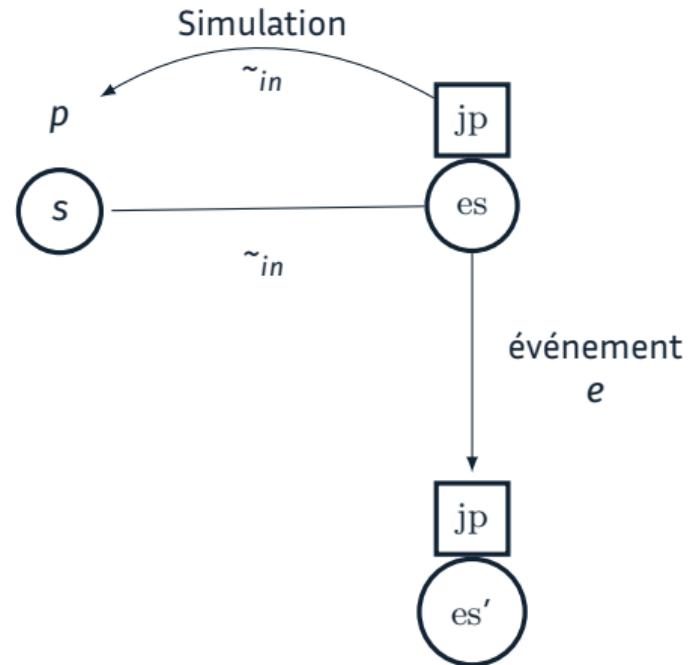
Spéculation vérifiée

- Définir la sémantique formelle des instructions spéculatives
- Prouver des simulations pour chaque étape (et plus encore : inlining,...).

Exécution (par ex. interprète) :
état d'exécution es mis à jour.

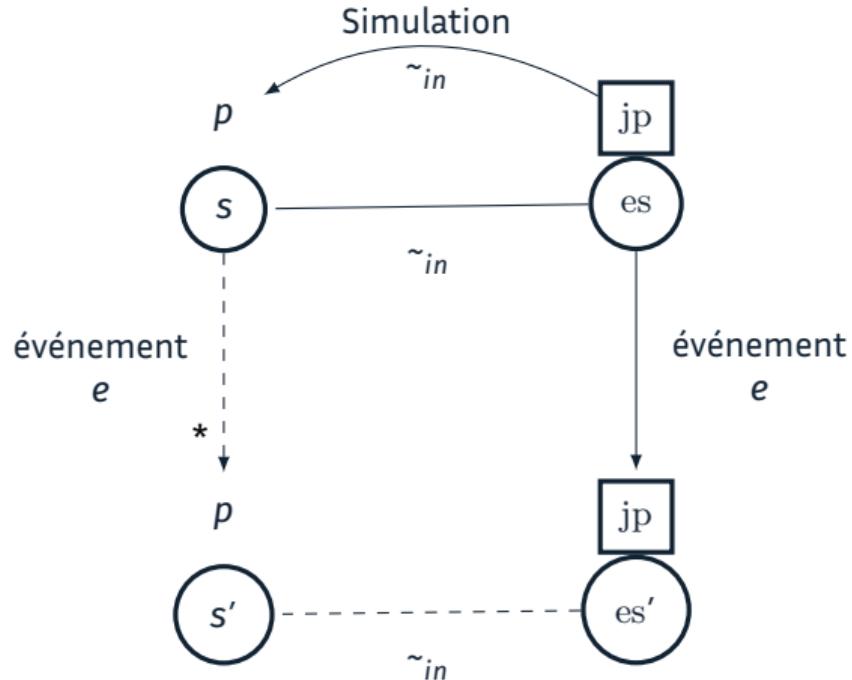


Exécution (par ex. interprète) :
état d'exécution es mis à jour.



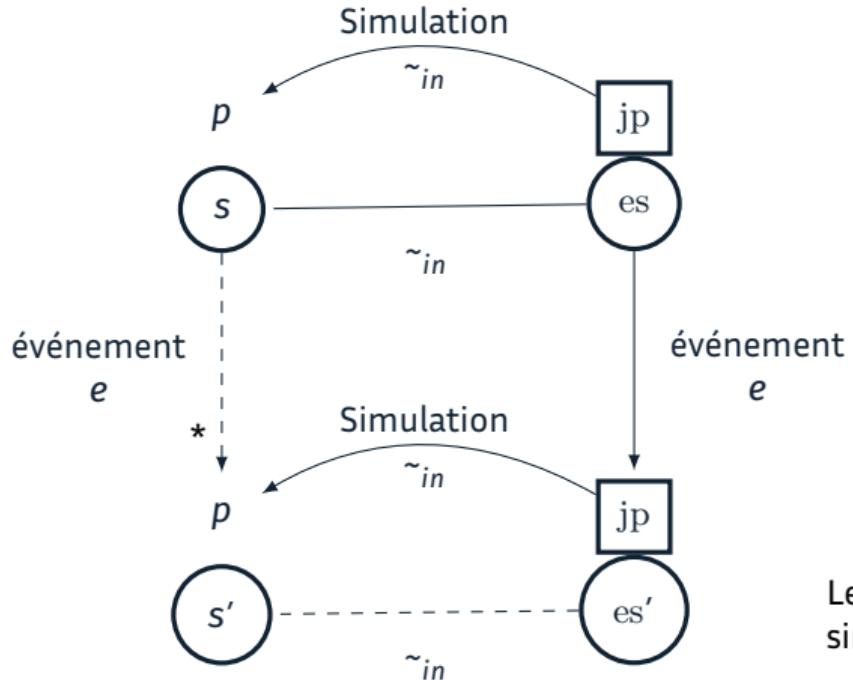
Exécution (par ex. interprète) :
état d'exécution es mis à jour.

On utilise la simulation interne
pour trouver une exécution
équivalente du source.



Exécution (par ex. interprète) :
état d'exécution es mis à jour.

On utilise la simulation interne
pour trouver une exécution
équivalente du source.



Le programme jp est toujours
simulé avec p.

(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

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

$$(\exists s'_1, s_1 \xrightarrow[p_1]{t} s'_1 \wedge s'_1 \sim_{int} s'_2) \text{ ou } (s_1 \sim_{int} s'_2 \wedge m_{int}(s'_2) < m_{int}(s_2) \wedge t = \emptyset)$$

interne

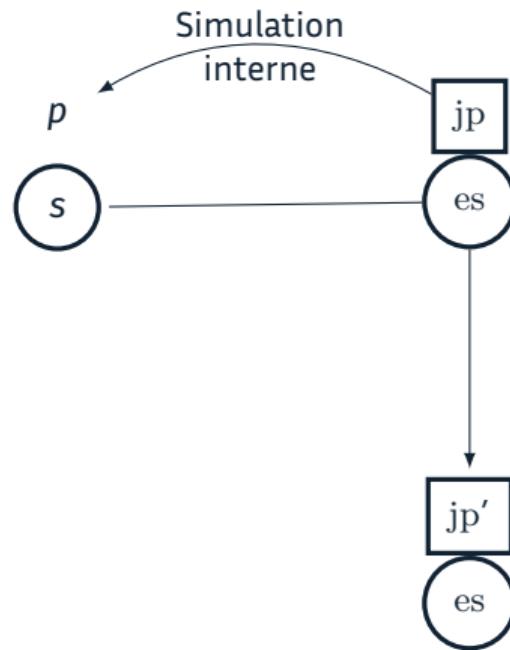
backward_internal_simulation $\sim_{int} m_{int} p_1 p$

 $s \sim_{int} e$ backward_internal_simulation $\sim_{int} m_{int} p_1 p$

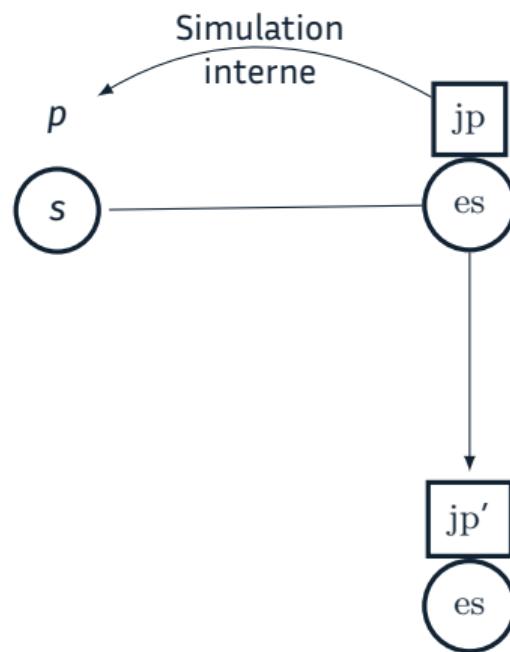
externe

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

Optimisation : programme jp
mis à jour.



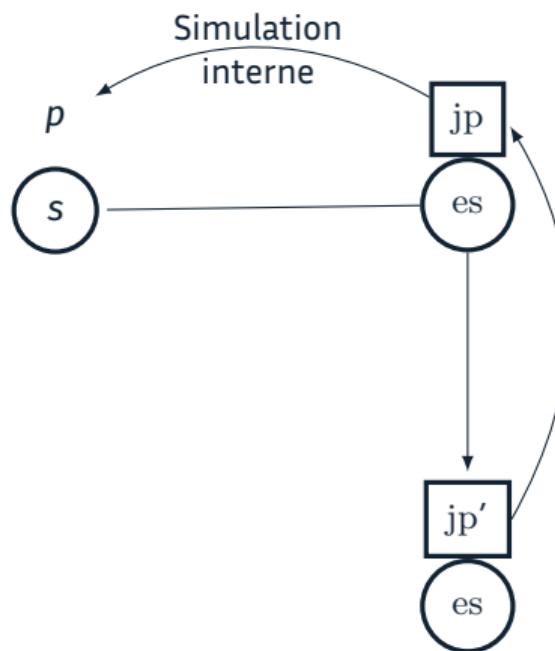
Optimisation : programme jp
mis à jour.



Il faut prouver que jp' est
toujours simulé avec p .

Optimisation : programme jp mis à jour.

On prouve l'optimisation dynamique correcte avec une simulation.

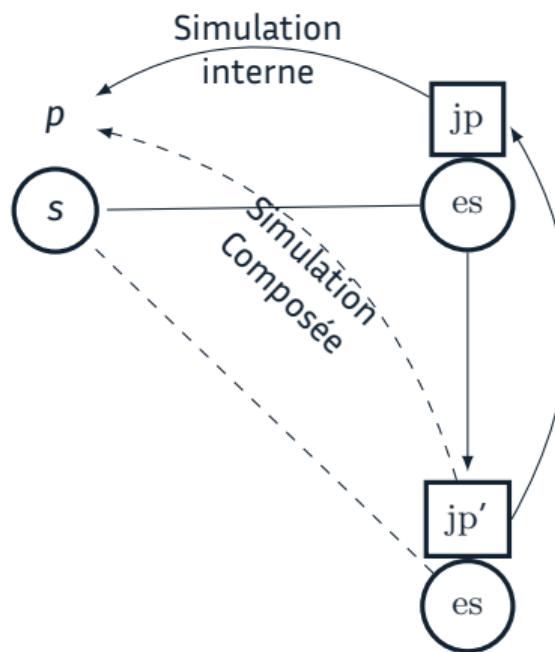


Il faut prouver que jp' est toujours simulé avec p .

Theorem optimizer_correct:
forall jp, jp',
optimizer jp = jp' →
backward_simulation jp jp'.

Optimisation : programme jp
mis à jour.

On prouve l'optimisation
dynamique correcte avec une
simulation.



Il faut prouver que jp' est
toujours simulé avec p .

Theorem optimizer_correct:
 $\forall jp\ jp',$
optimizer $jp = jp' \rightarrow$
backward_simulation $jp\ jp'$.

On compose cette simulation
avec la simulation interne
existante.

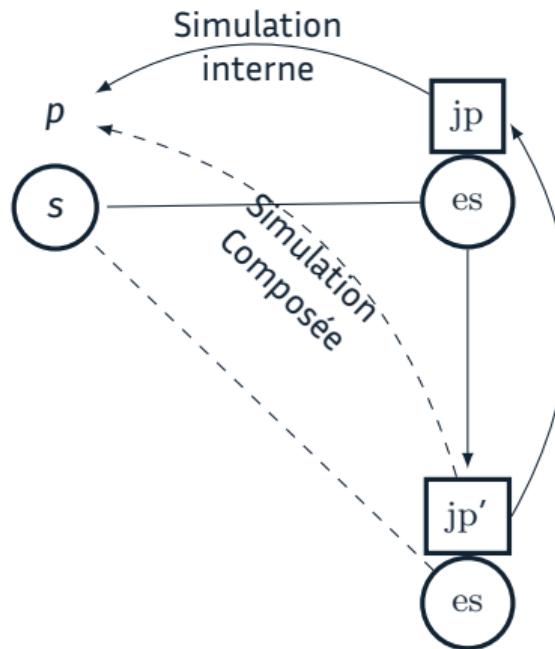
Optimisation : programme jp mis à jour.

On prouve l'optimisation dynamique correcte avec une simulation.

Conclusion

Avec cette technique, on prouve une optimisation dynamique avec une simulation, **comme dans le cas statique!**

Il faut prouver que jp' est toujours simulé avec p .



Theorem optimizer_correct:
 $\forall jp \, jp', \text{optimizer } jp = jp' \rightarrow \text{backward_simulation } jp \, jp'$

On compose cette simulation avec la simulation interne existante.

Matching ambigu

Dans les langages modernes (JavaScript, Python, PCRE, RE2, Rust), il faut retourner plus qu'un booléen.

- Match de $d \mid b$ sur "abcd" = "b".
- Match de $a \mid ab$ sur "abc" = "a".
- Match de ab^* sur "abbccc" = "abb".

Quel match choisir ?

Matching ambigu

Dans les langages modernes (JavaScript, Python, PCRE, RE2, Rust), il faut retourner plus qu'un booléen.

- Match de $d \mid b$ sur "abcd" = "b".
- Match de $a \mid ab$ sur "abc" = "a".
- Match de ab^* sur "abbccc" = "abb".

Quel match choisir ?

Règles de priorité

- Le match qui commence le plus tôt a la priorité.
- Dans $r_1 \mid r_2$, r_1 a la priorité (*non commutatif!*).
- Pour les quantificateurs *greedy* (*, +), priorité au nombre maximum d'itérations.

Matching ambigu

Dans les langages modernes (JavaScript, Python, PCRE, RE2, Rust), il faut retourner plus qu'un booléen.

- Match de $d \mid b$ sur "abcd" = "b".
- Match de $a \mid ab$ sur "abc" = "a".
- Match de ab^* sur "abbccc" = "abb".

Quel match choisir ?

Règles de priorité

- Le match qui commence le plus tôt a la priorité.
- Dans $r_1 \mid r_2$, r_1 a la priorité (*non commutatif!*).
- Pour les quantificateurs *greedy* (*, +), priorité au nombre maximum d'itérations.

Que faire pour ϵ^* ? La sémantique des quantificateurs doit éviter les répétitions infinies .

Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d'ε interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

(($a \mid \epsilon$) ($\epsilon \mid b$))^{*} on “ab”

Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

($(a \mid \epsilon)$ $(\epsilon \mid b)$) * on "ab"
 \xrightarrow{a}

Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

($(a \mid \epsilon)$ $(\epsilon \mid b)$) * on "ab"
 \xrightarrow{a} $\xrightarrow{\epsilon}$

Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

($(a \mid \epsilon)$ $(\epsilon \mid b)$) * on "ab"

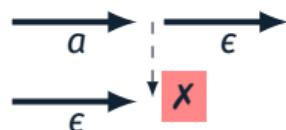


Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

($(a \mid \epsilon)$ $(\epsilon \mid b)$) * on "ab"

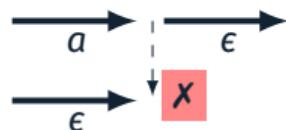


Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

($(a \mid \epsilon)$ $(\epsilon \mid b)$)* on "ab"



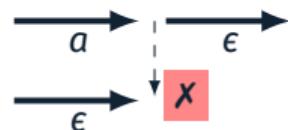
Résultat : 1 itération, matchant "a".

Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

($(a \mid \epsilon) \ (\epsilon \mid b)$) * on "ab"



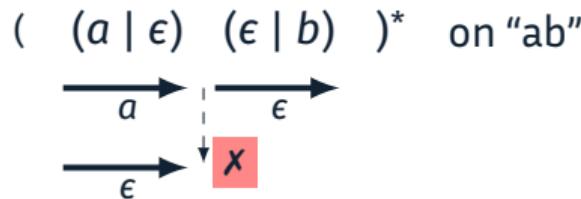
Résultat : 1 itération, matchant "a".

($(a \mid \epsilon) \ (\epsilon \mid b)$) * on "ab"

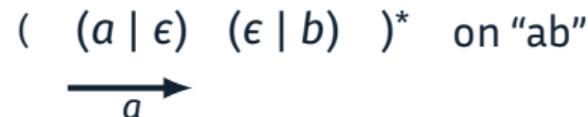
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..



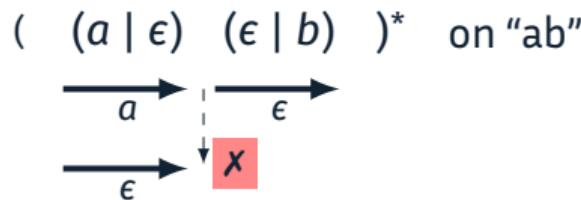
Résultat : 1 itération, matchant "a".



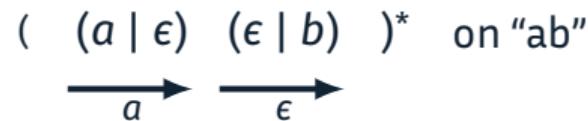
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..



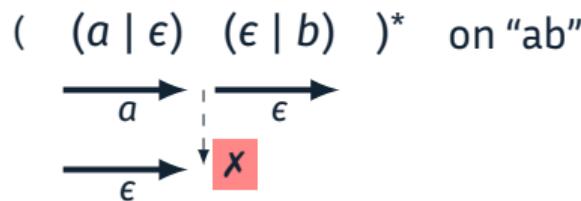
Résultat : 1 itération, matchant "a".



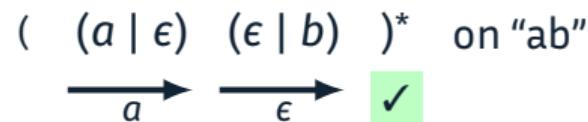
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..



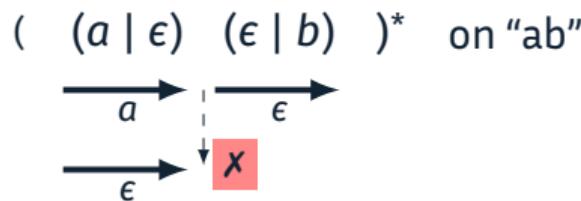
Résultat : 1 itération, matchant "a".



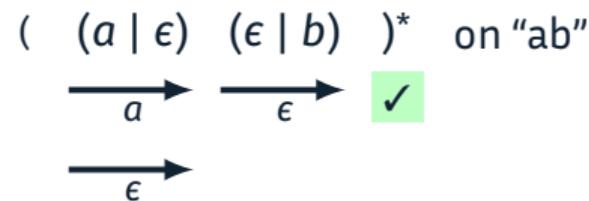
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..



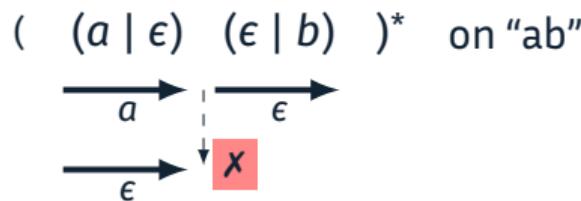
Résultat : 1 itération, matchant "a".



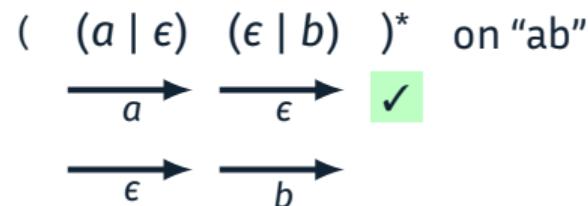
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..



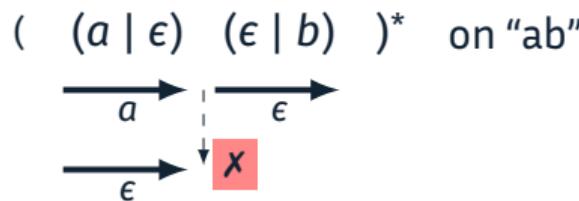
Résultat : 1 itération, matchant "a".



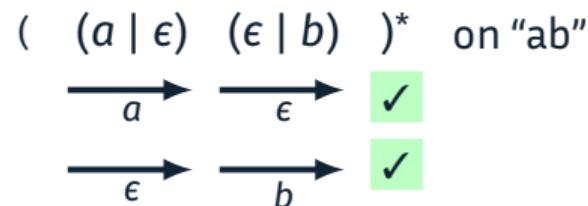
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.

JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..

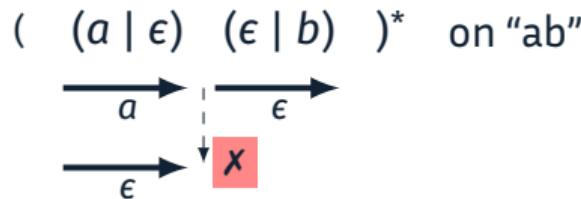


Résultat : 1 itération, matchant "a".



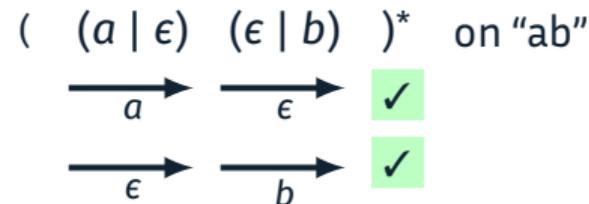
Deux manières d'éviter les boucles infinies

La majorité des langages : boucles d' ϵ interdites.



Résultat : 1 itération, matchant "a".

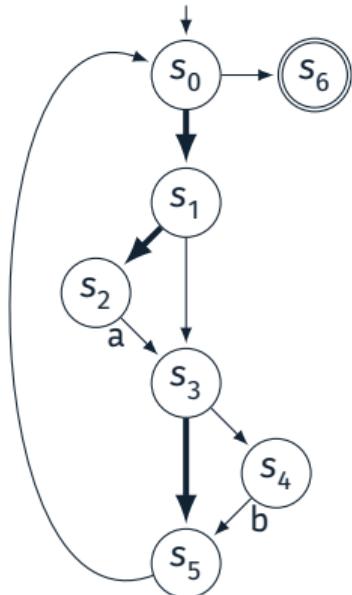
JavaScript : Les itérations ne peuvent pas matcher la chaîne vide..



Résultat : 2 itérations, matchant "ab".

Simulation de NFA sur "ab"

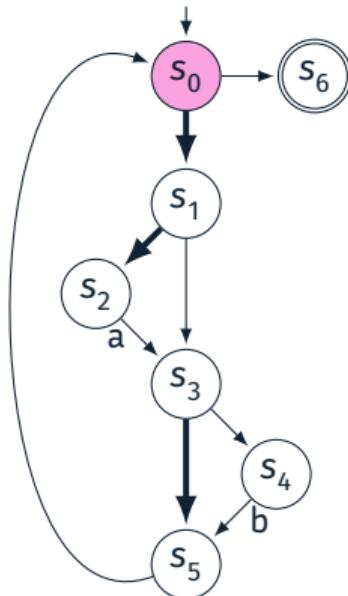
- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).



NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

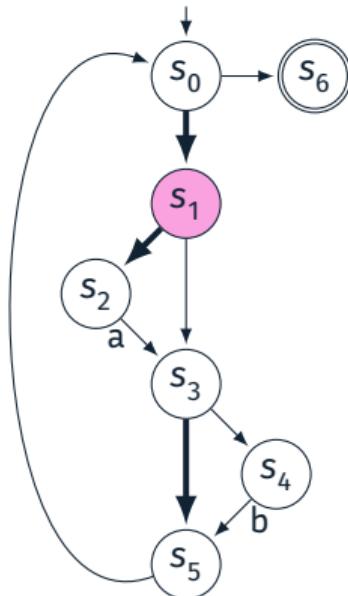


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

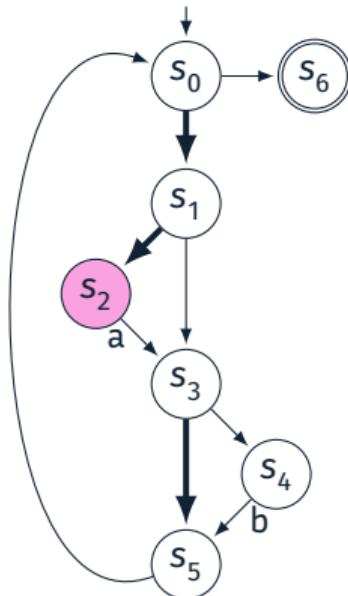


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

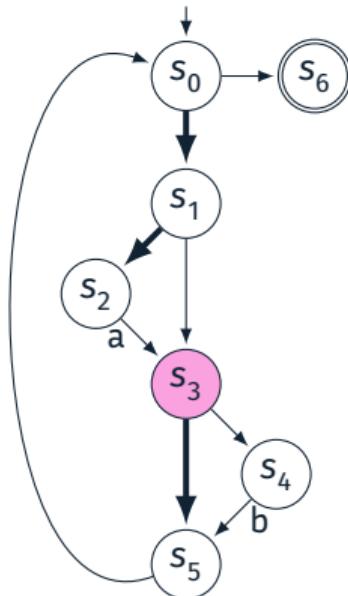


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

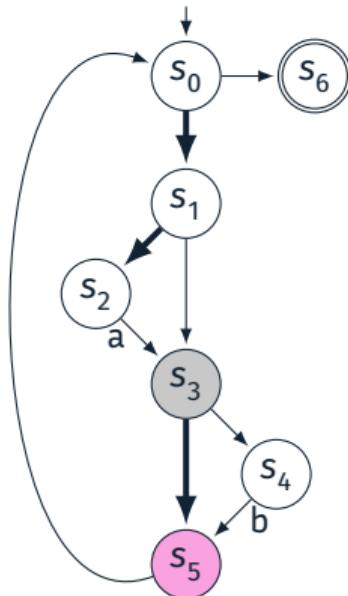


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

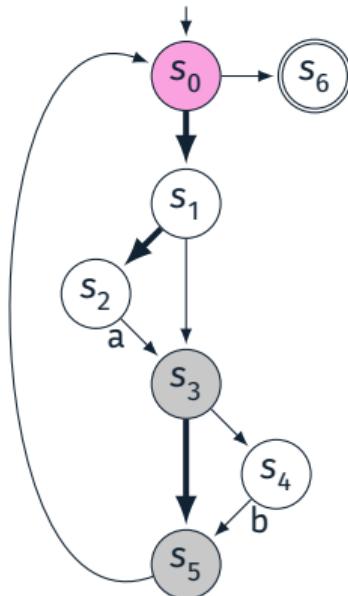


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

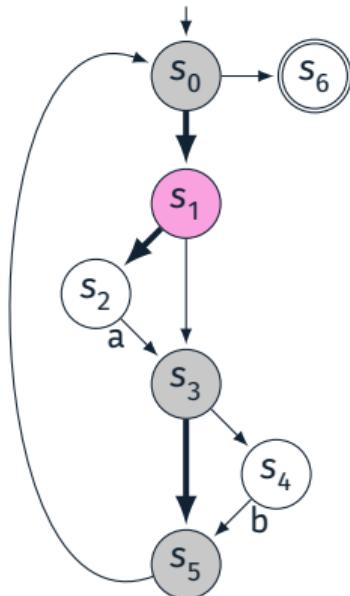


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

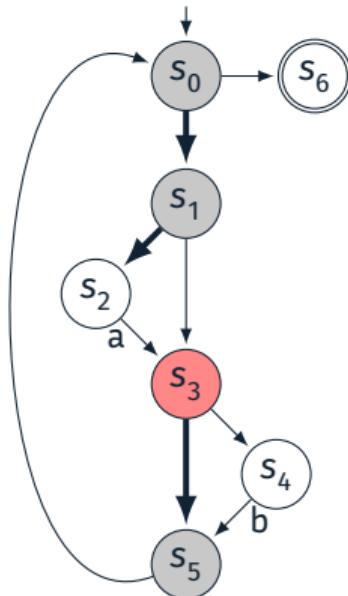


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Simulation de NFA sur "ab"

- Construire le NFA.
- Parcours en largeur pour trouver le chemin de plus haute priorité.
- Ne jamais visiter deux fois la même configuration (état du NFA + position de la chaîne).

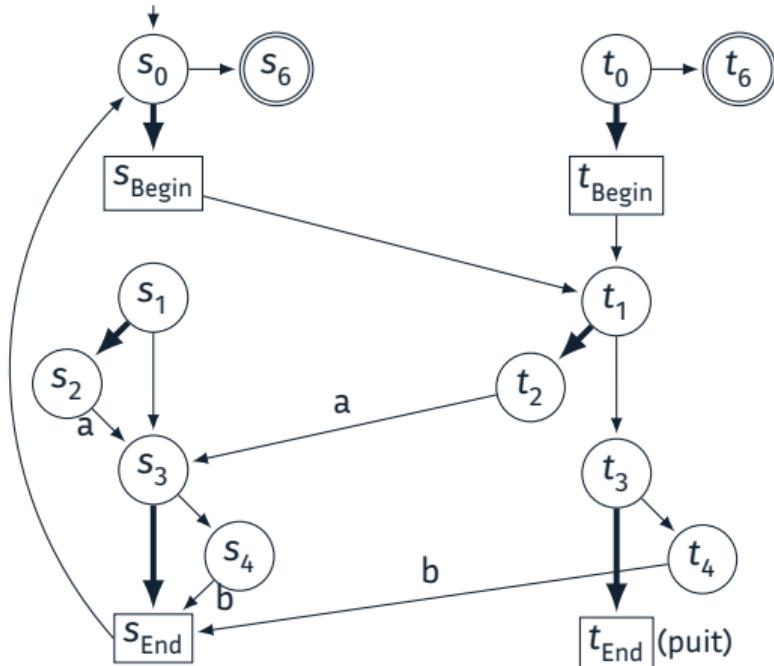


On ne trouve pas le chemin de plus haute priorité pour la sémantique JavaScript!

NFA de $((a \mid \epsilon) (\epsilon \mid b))^*$
avec arêtes de priorité en **gras**.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

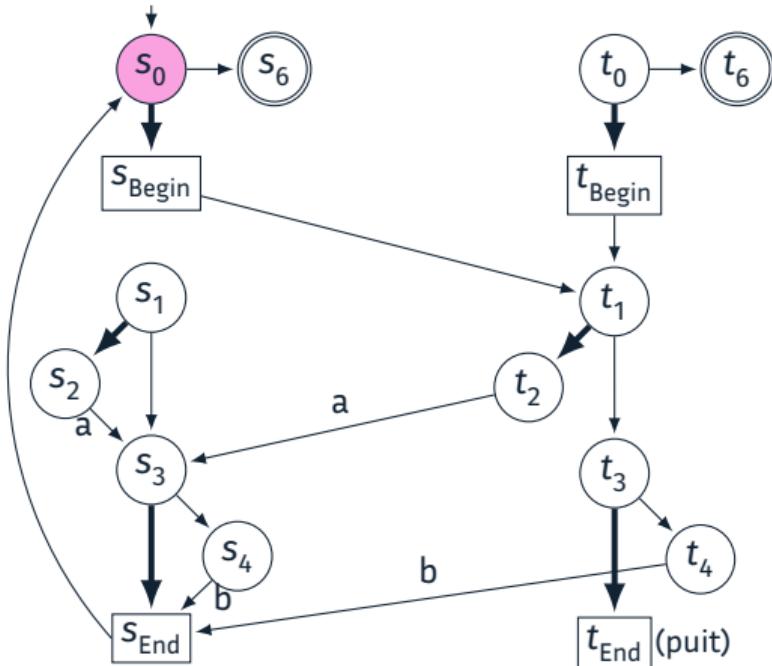


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

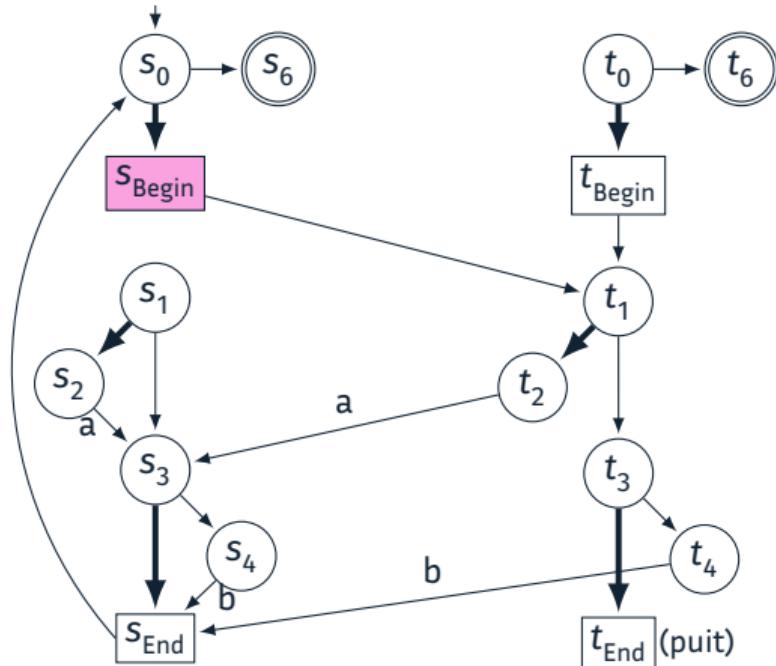


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

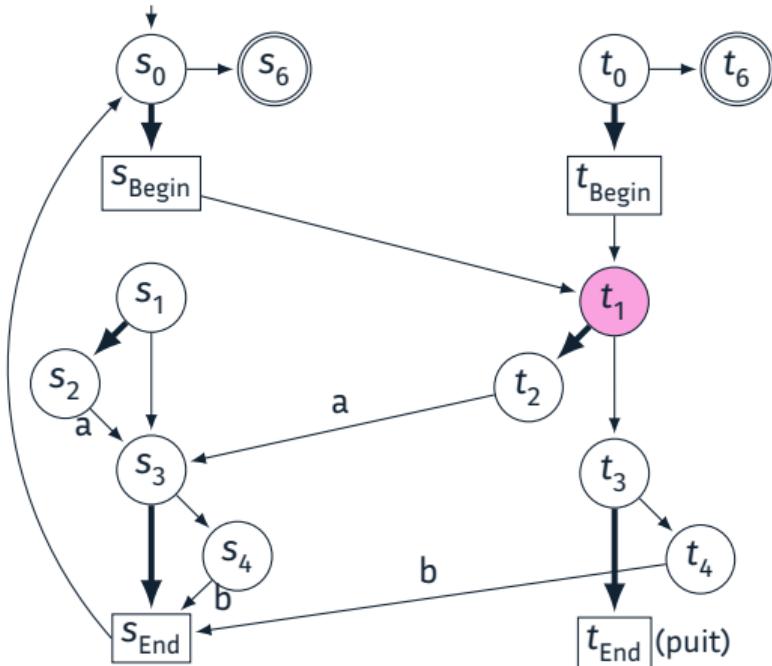


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

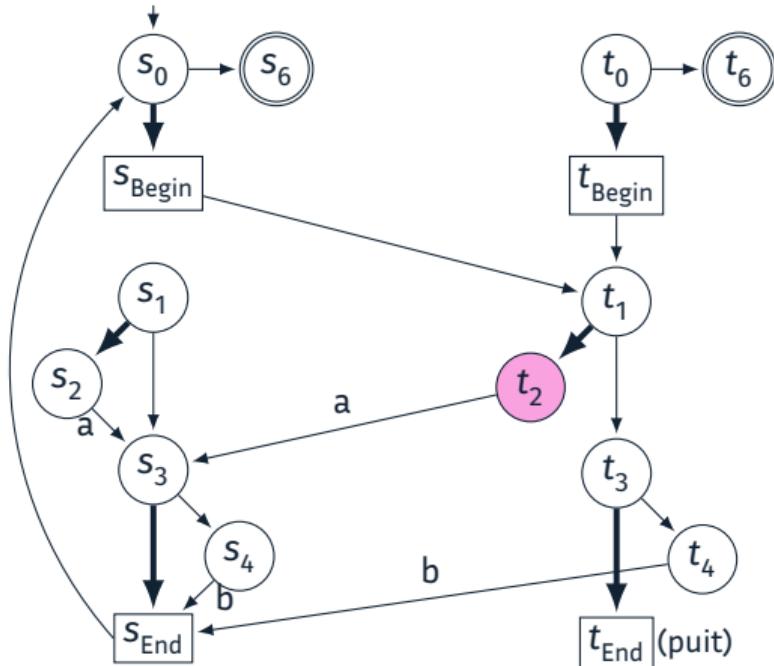


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

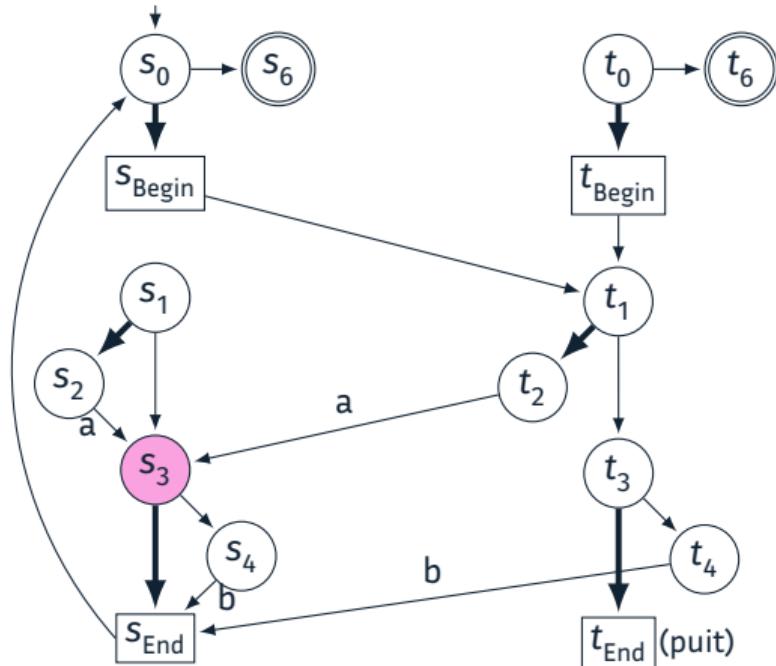


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

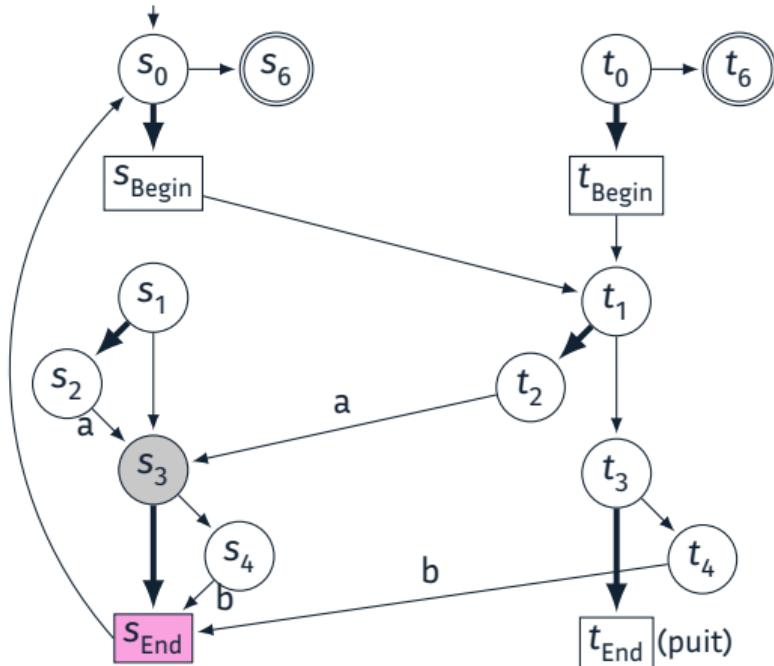


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

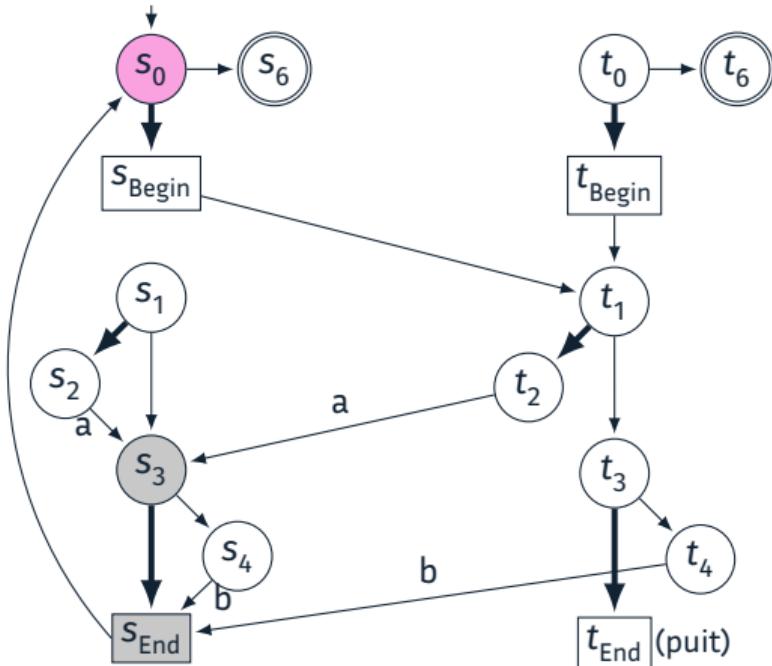


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

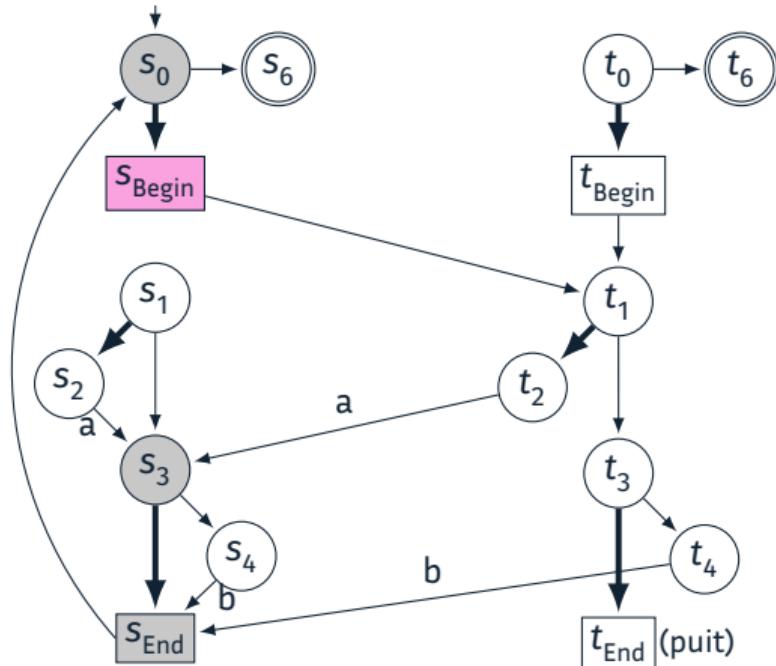


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

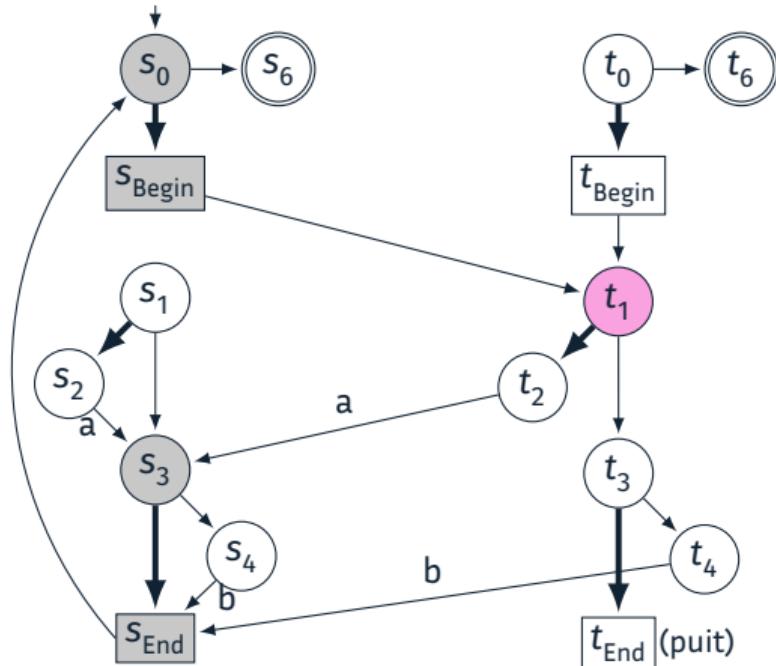


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

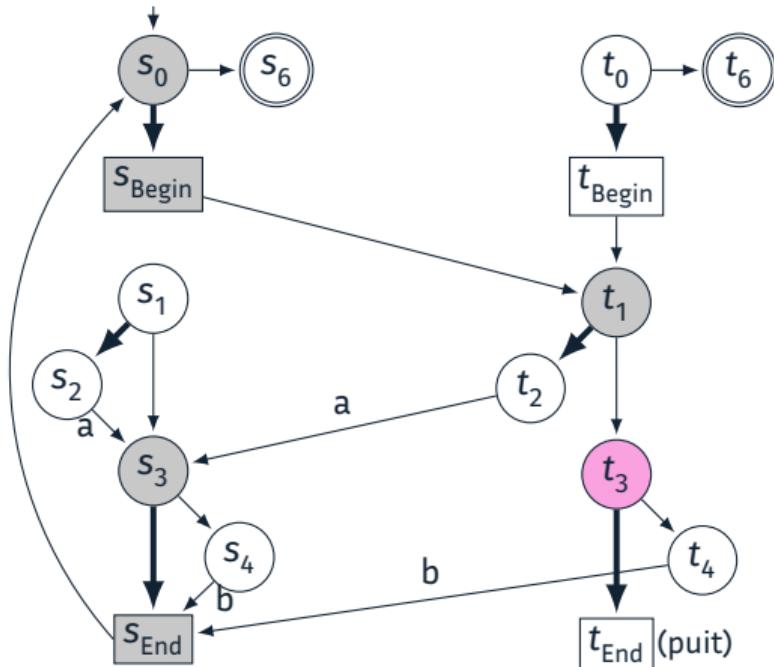


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

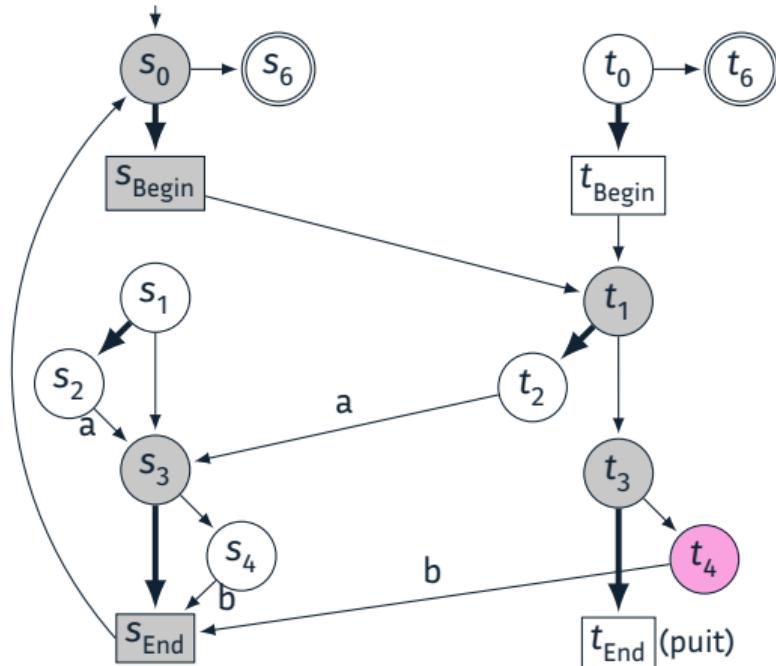


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

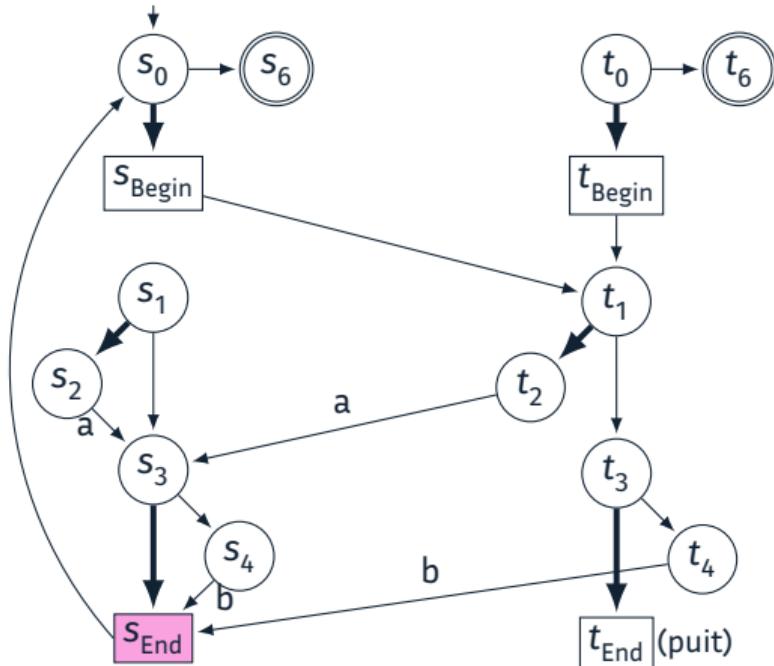


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

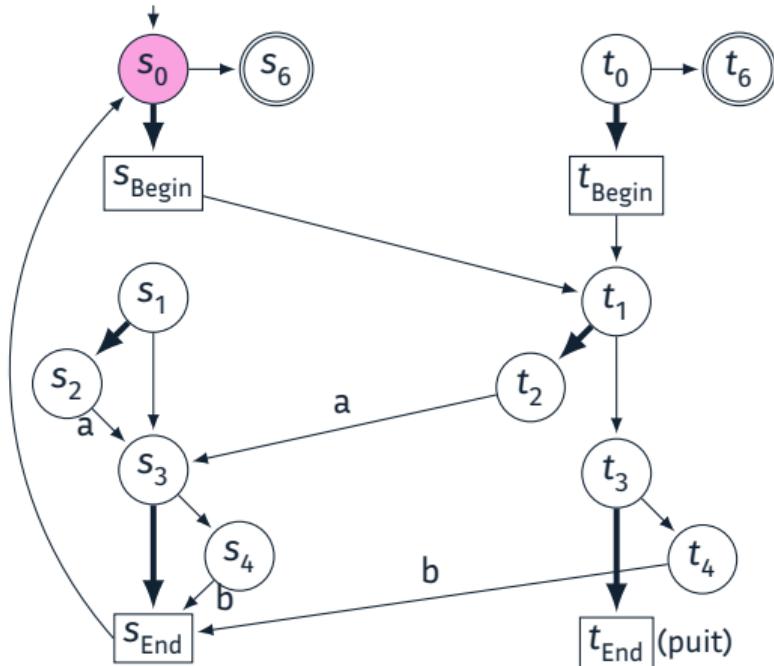


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

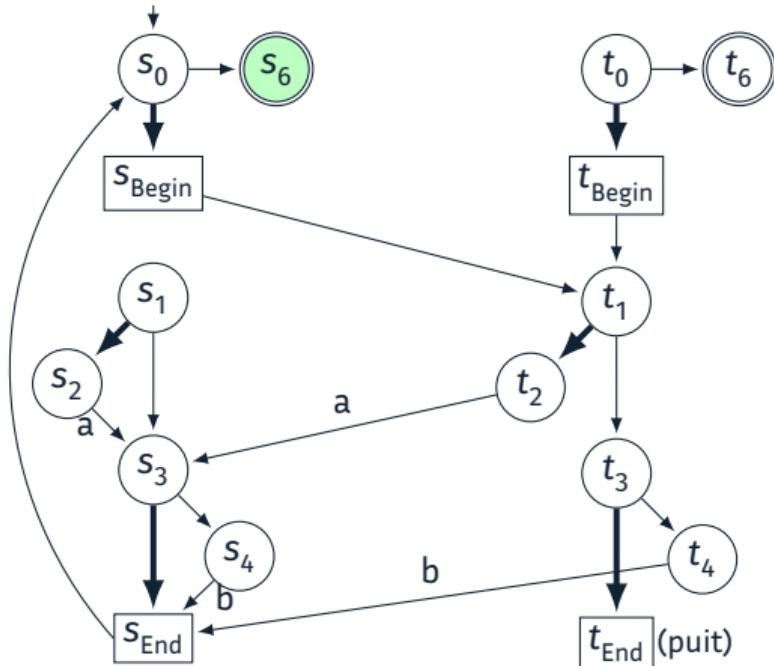


On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

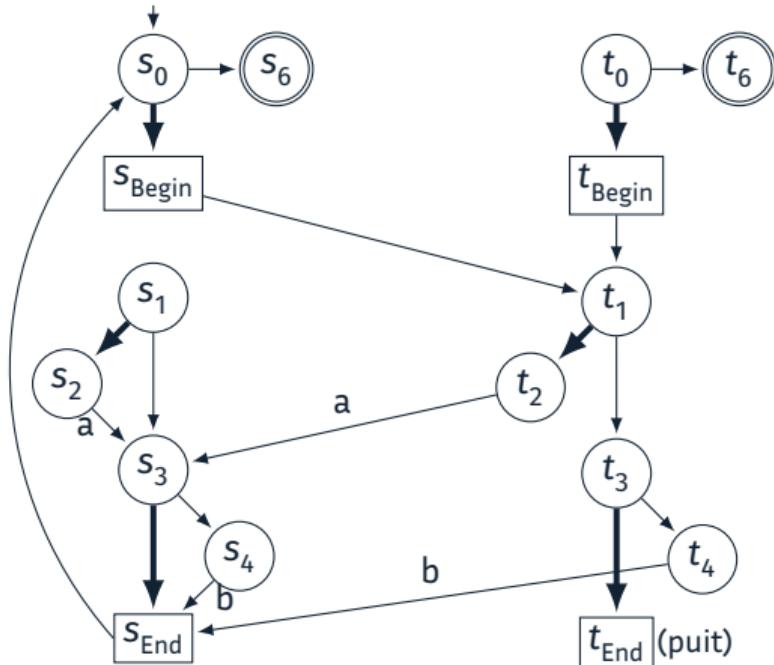
Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.



On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.



Nouvelle construction de NFA

- 2 copies du NFA.
- Nouveaux nœuds Begin et End.
- Begin pointe à droite.
- Lire un caractère pointe à gauche.

Algorithme linéaire pour l'étoile JavaScript

- Suit la sémantique JavaScript.
- Linéaire (deux copies seulement, même pour les étoiles imbriquées).
- Implémenté dans V8.

On peut sortir de l'étoile.

On ne peut pas sortir de l'étoile.

Groupes de capture

Retourner la sous-chaîne matchée en dernier par la sous-regex entre parenthèses.

Matcher $a(b \mid c)d$ sur "acde" = ("acd", "c").

Groupes de capture

Retourner la sous-chaîne matchée en dernier par la sous-regex entre parenthèses.

Matcher $a(b \mid c)d$ sur "acde" = ("acd", "c").

En JavaScript seulement : réinitialisation des captures

À chaque itération de l'étoile, réinitialiser les valeurs des groupes dans l'étoile.

Matcher $((a \mid b)^*$ sur "ab" = ("ab", "b", **undefined**).

Groupes de capture

Retourner la sous-chaîne matchée en dernier par la sous-regex entre parenthèses.

Matcher $a(b \mid c)d$ sur "acde" = ("acd", "c").

En JavaScript seulement : réinitialisation des captures

À chaque itération de l'étoile, réinitialiser les valeurs des groupes dans l'étoile.

Matcher $((a \mid b)^*$ sur "ab" = ("ab", "b", **undefined**).

Lookarounds

Une condition dans une regex. $a(?=b)$ matche les "a", seulement si ils sont suivis d'un "b".

$[0-9]^+(?=^\circ C)$ matche "12" dans "12°C", mais rien dans "12 mars".

Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookaround est vrai.

En *inversant* la regex.

Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookaround est vrai.

En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.

Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c
(a b)	✓	✓	✓	✗
				✗

Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

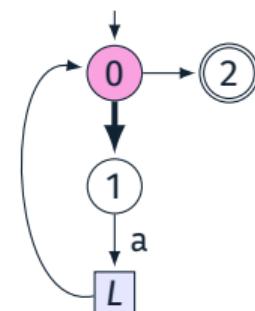
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

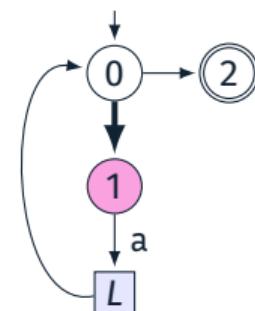
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

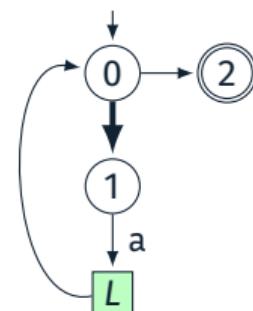
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c
(a b)	✓	✓	✓	x
				x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

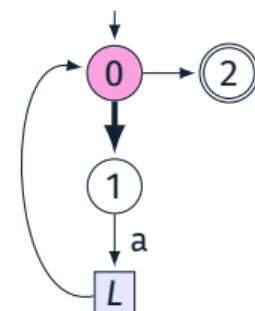
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

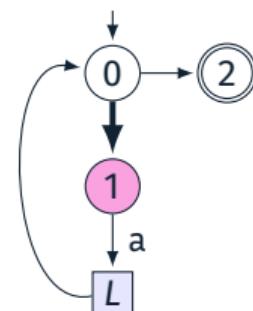
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c
(a b)	✓	✓	✓	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

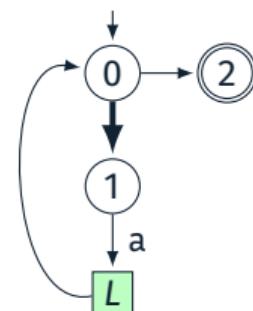
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c
(a b)	✓	✓	✓	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

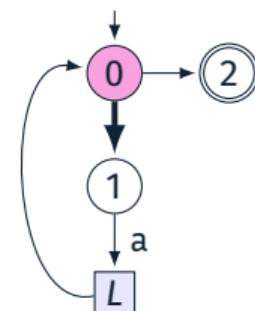
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

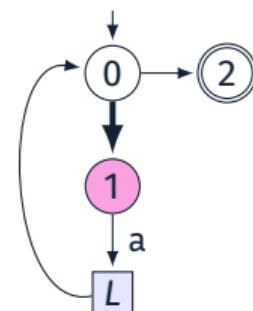
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

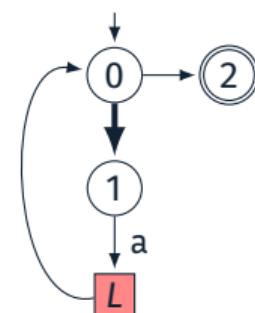
En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c
(a b)	✓	✓	✓	X



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

En *inversant* la regex.

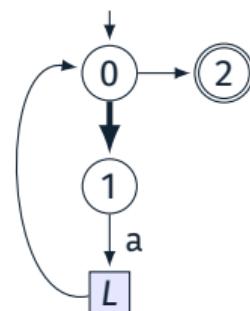
Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Et si les lookarounds ont des groupes?

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

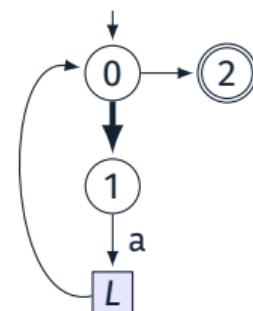
Et si les lookaheads ont des groupes?

Idée clé 2 - Conséquence de la réinitialisation

Chaque groupe dans un lookahead ne peut être défini que par le dernier usage du lookahead.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Et si les lookaheads ont des groupes?

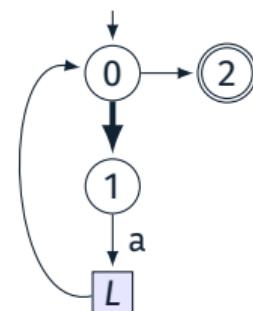
- Reconstruire les groupes des lookahead :
Matcher chaque lookahead une fois à partir de leur dernier usage.

Idée clé 2 - Conséquence de la réinitialisation

Chaque groupe dans un lookahead ne peut être défini que par le dernier usage du lookahead.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c	
(a b)	✓	✓	✓	x	x



Idée clé 1 - Précalcul

En temps linéaire, on peut précalculer chaque position où un lookahead est vrai.

En *inversant* la regex.

Un algorithme en 3 étapes

- Précalculer une table.
- Matcher l'expression principale.

Et si les lookarounds ont des groupes?

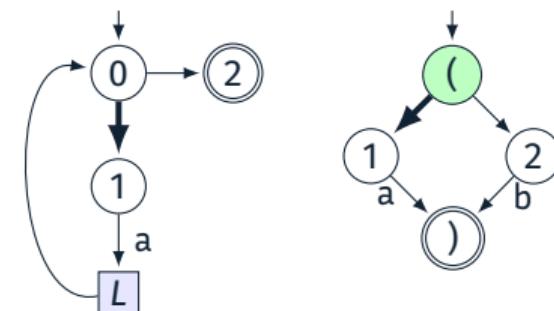
- Reconstruire les groupes des lookahead :
Matcher chaque lookahead une fois à partir de leur dernier usage.

Idée clé 2 - Conséquence de la réinitialisation

Chaque groupe dans un lookahead ne peut être défini que par le dernier usage du lookahead.

Exemple : $(a \ (?=(a | b)))^*$ sur "aaac".

	a	a	a	c
(a b)	✓	✓	✓	x



22.2.2.4.1 `IsWordChar(rer, Input, e)`

The abstract operation `IsWordChar` takes arguments `rer` (a `RegExp Record`), `Input` (a `List` of characters), and `e` (an `integer`) and returns a `Boolean`.

It performs the following steps when called:

1. Let `InputLength` be the number of elements in `Input`.
2. If `e = -1` or `e = InputLength`, return `false`.
3. Let `c` be the character `Input[e]`.
4. If `WordCharacters(rer)` contains `c`, return `true`.
5. Return `false`.

```
(** >>
 22.2.2.4.1 IsWordChar ( rer, Input, e )
```

The abstract operation IsWordChar takes arguments rer (a RegExp Record), Input (a List of characters), and e (an integer) and returns a Boolean.

It performs the following steps when called:

```
<<*>
```

(*>> 1. Let InputLength be the number of elements in Input. <<*)

(*>> 2. If $e = -1$ or $e = \text{InputLength}$, return false. <<*)

(*>> 3. Let c be the character $\text{Input}[e]$. <<*)

(*>> 4. If WordCharacters(rer) contains c , return true. <<*)

(*>> 5. Return false. <<*)

```
(** >>
 22.2.2.4.1 IsWordChar ( rer, Input, e )
```

The abstract operation IsWordChar takes arguments rer (a RegExp Record), Input (a List of characters), and e (an integer) and returns a Boolean.

It performs the following steps when called:

```
<<*>
Definition isWordChar(rer:RegExpRecord)(Input:list Character)(e:integer):Result bool :=
  (*>> 1. Let InputLength be the number of elements in Input. <<*)
  let InputLength:=List.length Input in
  (*>> 2. If e = -1 or e = InputLength, return false. <<*)
  if (e =? -1)%Z || (e =? InputLength)%Z then false
  else
    (*>> 3. Let c be the character Input[ e ]. <<*)
    let! c =<< Input[e] in
    (*>> 4. If WordCharacters(rer) contains c, return true. <<*)
    let! wc =<< wordCharacters rer in
    if CharSet.contains wc c then true
    else
      (*>> 5. Return false. <<*)
      false.
```

1/16 Mon domaine

2/16 Compilation formellement vérifiée

3/16 Écosystème web, compilation non traditionnelle

4/16 Ma méthodologie

5/16 Doctorat

6/16 Spéculation & Déoptimisation

7/16 Simulations imbriquées

8/16 JITs vérifiés

9/16 PostDoc

10/16 Nouveaux algorithmes linéaires

11/16 Nouvelle sémantique des regex

12/16 Avancements algorithmiques et sémantiques

13/16 Vers une plateforme web de confiance

14/16 Un moteur vérifié, linéaire, efficace, intégrable

15/16 Un sous-ensemble WebAssembly de confiance

16/16 Vérification formelle pour un Web de confiance

Transparents supplémentaires :

Spéculer dans un langage dynamique

Insérer des instructions spéculatives

Définition Simulations Imbriquées

Simulations imbriquées, exécution

Simulations imbriquées, optimisation

Regex modernes avec priorité

L'étoile JavaScript est unique

Simulation de NFA et étoile

Dupliquer le graphe pour l'étoile JavaScript

Lookarounds et groupes de capture

3 étapes pour les lookarounds

Une mécanisation de confiance