Présentation de WebAssembly

Léo Andrès

23 avril 2021

OCaml PRO

JavaScript

- créé en 1995…

JavaScript

- créé en 1995…
- … en 10 jours

JavaScript

- créé en 1995…
- … en 10 jours
- seul moyen d'avoir des pages web interactives aujourd'hui

JavaScript

propice aux bugs :

```
[] == ![]; // -> true
Number.MIN_VALUE > 0; // -> true
parseInt("firetruck"); // -> NaN
parseInt("firetruck", 16); // -> 15
Math.min() > Math.max(); // -> true
```

repris d'une présentation de Guy Royse (Redis Labs)

JavaScript

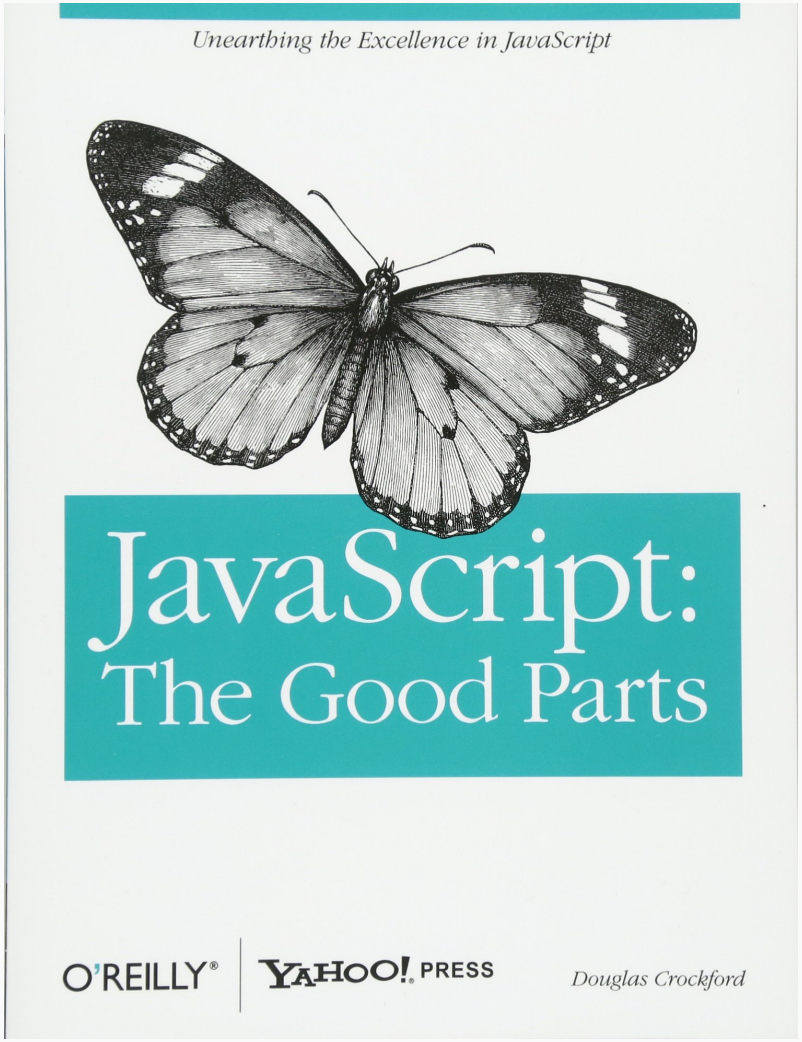
propice aux bugs :

```
[] == ![]; // -> true
Number.MIN_VALUE > 0; // -> true
parseInt("firetruck"); // -> NaN
parseInt("firetruck", 16); // -> 15
Math.min() > Math.max(); // -> true
```

repris d'une présentation de Guy Royse (Redis Labs)

liste plus complète : github.com/denysdovhan/wtfjs

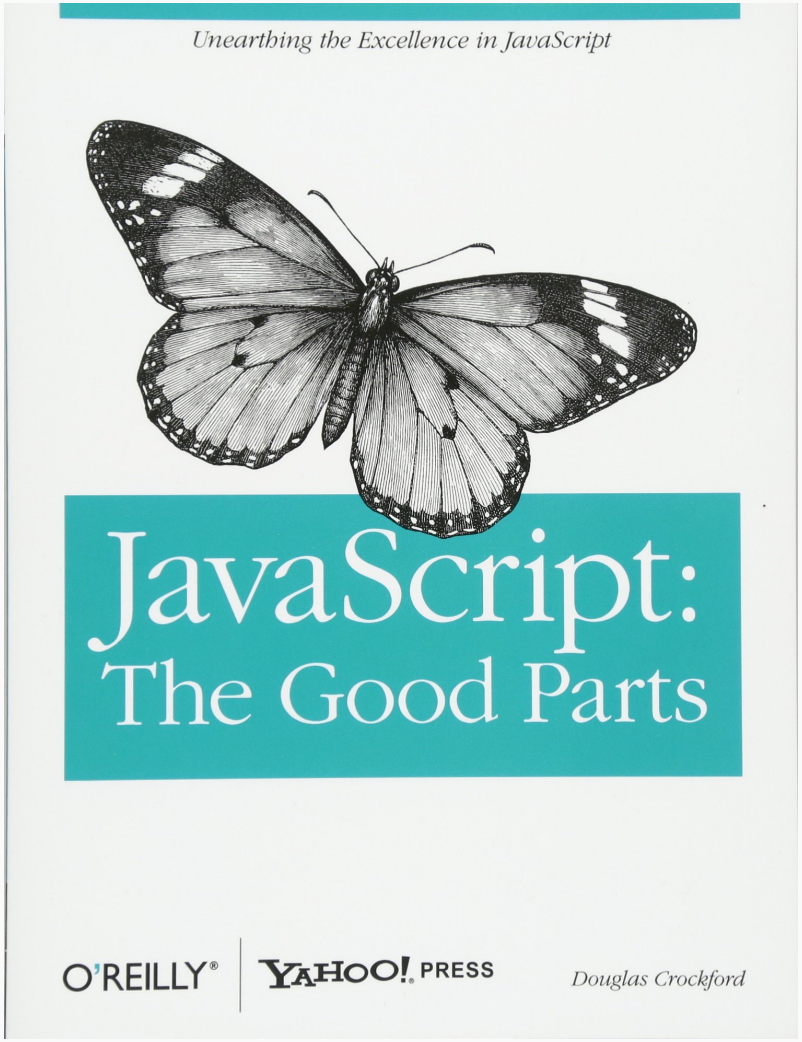
Alternatives à JavaScript



(2008)

- se restreindre à un sous-ensemble du langage

Alternatives à JavaScript



(2008)

- se restreindre à un sous-ensemble du langage
- ne règle qu'une partie des problèmes

Alternatives à JavaScript

- utiliser un sur-ensemble du langage : TypeScript (2012), flow (2014)

Alternatives à JavaScript

- utiliser un sur-ensemble du langage : TypeScript (2012), flow (2014)
- ne règle qu'une partie des problèmes

Alternatives à JavaScript

- compiler son langage favori vers JavaScript : js_of_ocaml (2010)

Alternatives à JavaScript

- compiler son langage favori vers JavaScript : js_of_ocaml (2010)
- ne règle qu'une partie des problèmes

Alternatives à JavaScript

- compiler son langage favori vers JavaScript : js_of_ocaml (2010)
- ne règle qu'une partie des problèmes
- difficilement imposable partout

Alternatives à JavaScript

(2016)

- garder la syntaxe JavaScript

Alternatives à JavaScript

(2016)

- garder la syntaxe JavaScript
- l'utiliser comme syntaxe alternative dans OCaml

Alternatives à JavaScript

(2016)

- garder la syntaxe JavaScript
- l'utiliser comme syntaxe alternative dans OCaml
- compiler OCaml vers JavaScript

Alternatives à JavaScript

(2016)

- garder la syntaxe JavaScript
- l'utiliser comme syntaxe alternative dans OCaml
- compiler OCaml vers JavaScript
- ne règle qu'une partie des problèmes

asm.js

(2013)

- sous-ensemble de JavaScript

asm.js

(2013)

- sous-ensemble de JavaScript
- optimisé pour compiler du C efficacement

asm.js

(2013)

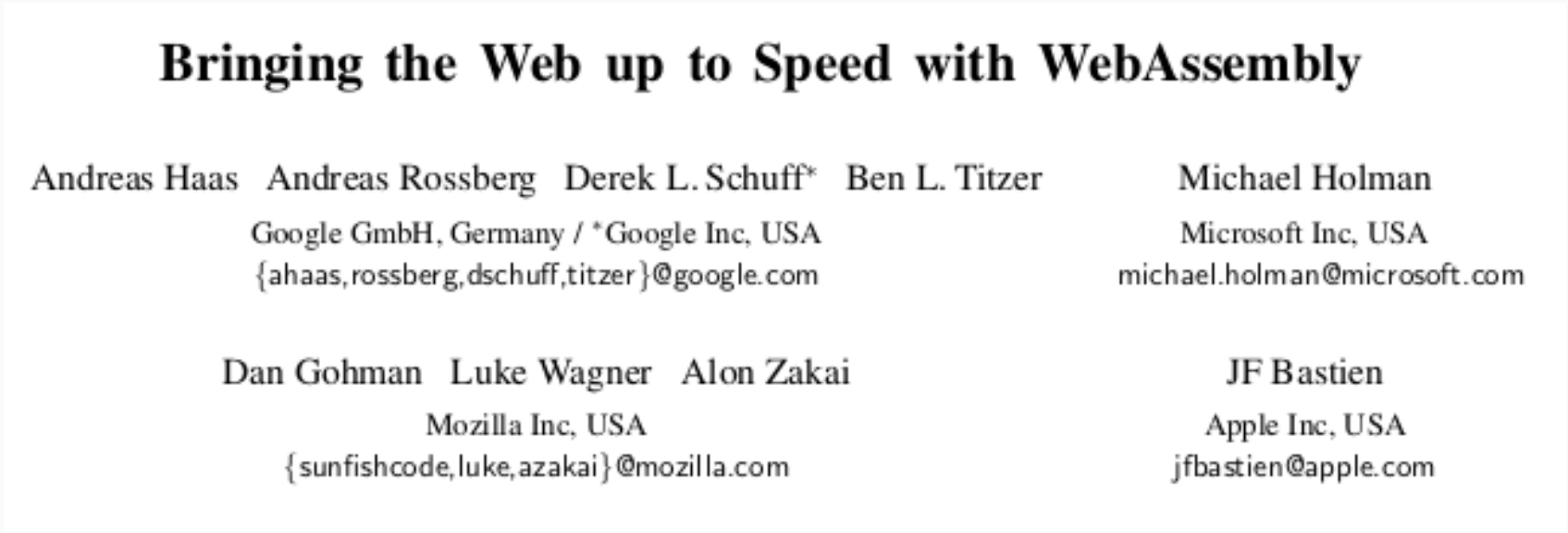
- sous-ensemble de JavaScript
- optimisé pour compiler du C efficacement
- succès d'Emscripten (Unreal Engine porté sur firefox)

asm.js

(2013)

- sous-ensemble de JavaScript
- optimisé pour compiler du C efficacement
- succès d'Emscripten (Unreal Engine porté sur firefox)
- restreint aux langages compilés vers du bitcode LLVM

WebAssembly



Bringing the Web up to Speed with WebAssembly

Andreas Haas Andreas Rossberg Derek L. Schuff* Ben L. Titzer
Google GmbH, Germany / *Google Inc, USA
{ahaas,rossberg,dschuff,titzer}@google.com

Michael Holman
Microsoft Inc, USA
michael.holman@microsoft.com

Dan Gohman Luke Wagner Alon Zakai
Mozilla Inc, USA
{sunfishcode,luke,azakai}@mozilla.com

JF Bastien
Apple Inc, USA
jfbastien@apple.com

PLDI 2017

WebAssembly : objectifs

Un langage cible bas-niveau :

- rapide

WebAssembly : objectifs

Un langage cible bas-niveau :

- rapide
- sûr

WebAssembly : objectifs

Un langage cible bas-niveau :

- rapide
- sûr
- portable

WebAssembly : objectifs

Une sémantique sûre, rapide et portable :

- sûre à l'exécution

WebAssembly : objectifs

Une sémantique sûre, rapide et portable :

- sûre à l'exécution
- rapide à l'exécution

WebAssembly : objectifs

Une sémantique sûre, rapide et portable :

- sûre à l'exécution
- rapide à l'exécution
- indépendante du langage, du matériel et de la plate-forme

WebAssembly : objectifs

Une sémantique sûre, rapide et portable :

- sûre à l'exécution
- rapide à l'exécution
- indépendante du langage, du matériel et de la plate-forme
- déterministe et permettant un raisonnement facile

WebAssembly : objectifs

Une sémantique sûre, rapide et portable :

- sûre à l'exécution
- rapide à l'exécution
- indépendante du langage, du matériel et de la plate-forme
- déterministe et permettant un raisonnement facile
- inter-opérabilité simple avec le Web

WebAssembly : objectifs

Une représentation sûre et efficiente :

- compacte et facile à décoder

WebAssembly : objectifs

Une représentation sûre et efficiente :

- compacte et facile à décoder
- facile à valider et compiler

WebAssembly : objectifs

Une représentation sûre et efficiente :

- compacte et facile à décoder
- facile à valider et compiler
- facile à générer

WebAssembly : objectifs

Une représentation sûre et efficiente :

- compacte et facile à décoder
- facile à valider et compiler
- facile à générer
- streamable et parallélisable

Sûreté

- on ne peut pas faire confiance au code

Sûreté

- on ne peut pas faire confiance au code
- besoin d'un environnement d'exécution contrôlé (VM JavaScript)

Sûreté

- on ne peut pas faire confiance au code
- besoin d'un environnement d'exécution contrôlé (VM JavaScript)
- sûreté mémoire : impossible de compromettre les données et le système

Sûreté

- on ne peut pas faire confiance au code
- besoin d'un environnement d'exécution contrôlé (VM JavaScript)
- sûreté mémoire : impossible de compromettre les données et le système
- pas adapté à des applications C/C++ intrinsèquement rapides, sans GC et sans sûreté mémoire

Rapidité

- code C/C++ optimisé à la compilation

Rapidité

- code C/C++ optimisé à la compilation
- peut tirer partie des spécificités de la machine

Rapidité

- code C/C++ optimisé à la compilation
- peut tirer partie des spécificités de la machine
- environnement d'exécution type VM impose une dégradation de performance importante

Portabilité

- le Web est utilisé sur de nombreux appareils, architectures, OS et navigateurs

Portabilité

- le Web est utilisé sur de nombreux appareils, architectures, OS et navigateurs
- le code doit être indépendant du matériel et de la plate-forme pour garantir le même comportement

Portabilité

- le Web est utilisé sur de nombreux appareils, architectures, OS et navigateurs
- le code doit être indépendant du matériel et de la plate-forme pour garantir le même comportement
- tentatives passées de langage bas niveau : liées à une seule architecture ou problèmes de portabilité

Compacité

- code transmis sur le réseau doit être compact

Compacité

- code transmis sur le réseau doit être compact
- réduction des temps de chargements, économie de bande passante, bonne réactivité

Compacité

- code transmis sur le réseau doit être compact
- réduction des temps de chargements, économie de bande passante, bonne réactivité
- le code JS, même minifié et compressé est bien moins compact qu'un format binaire

Modules

(modules) définitions de fonctions, globales, tables et mémoires :

$$\mathrm{m} ::= \texttt{module}\ \mathrm{f}^*\ \mathrm{glob}^*\ \mathrm{tab}^?\ \mathrm{mem}^?$$

Modules

(modules) définitions de fonctions, globales, tables et mémoires :

$$\mathrm{m} ::= \texttt{module}\ \mathrm{f}^{*}\ \mathit{glob}^{*}\ \mathit{tab}^{?}\ \mathit{mem}^{?}$$

(exports) définition peut être exportée plusieurs fois :

$$\mathit{ex} ::= \texttt{export}\ \text{"}\mathit{name}\text{"}$$

Modules

(modules) définitions de fonctions, globales, tables et mémoires :

$$\mathrm{m} ::= \texttt{module}\ \mathrm{f}^*\ glob^*\ tab^?\ mem^?$$

(exports) définition peut être exportée plusieurs fois :

$$ex ::= \texttt{export}\ \text{"}name\text{"}$$

(imports) définition importée via module, item et type :

$$\mathrm{im} ::= \texttt{import}\ \text{"}name\text{"}\ \text{"}name\text{"}$$

Modules

(modules) définitions de fonctions, globales, tables et mémoires :

$$\mathrm{m} ::= \texttt{module}\ \mathrm{f}^*\ \mathrm{glob}^*\ \mathrm{tab}^?\ \mathrm{mem}^?$$

(exports) définition peut être exportée plusieurs fois :

$$\mathit{ex} ::= \texttt{export}\ \text{"}\mathrm{name}\text{"}$$

(imports) définition importée via module, item et type :

$$\mathrm{im} ::= \texttt{import}\ \text{"}\mathrm{name}\text{"}\ \text{"}\mathrm{name}\text{"}$$

- modules doivent être instanciés par l'*embedder* (JS, OS)

Modules

(modules) définitions de fonctions, globales, tables et mémoires :

$$\mathrm{m} ::= \texttt{module}\ \mathrm{f}^*\ \mathrm{glob}^*\ \mathrm{tab}^?\ \mathrm{mem}^?$$

(exports) définition peut être exportée plusieurs fois :

$$\mathit{ex} ::= \texttt{export}\ \text{"}\mathrm{name}\text{"}$$

(imports) définition importée via module, item et type :

$$\mathrm{im} ::= \texttt{import}\ \text{"}\mathrm{name}\text{"}\ \text{"}\mathrm{name}\text{"}$$

- modules doivent être instanciés par l'*embedder* (JS, OS)
- on doit fournir des définitions pour tous les *imports*

Modules

(modules) définitions de fonctions, globales, tables et mémoires :

$$\mathrm{m} ::= \texttt{module}\ \mathrm{f}^*\ \mathit{glob}^*\ \mathit{tab}^?\ \mathit{mem}^?$$

(exports) définition peut être exportée plusieurs fois :

$$\mathit{ex} ::= \texttt{export}\ \text{"}\mathit{name}\text{"}$$

(imports) définition importée via module, item et type :

$$\mathrm{im} ::= \texttt{import}\ \text{"}\mathit{name}\text{"}\ \text{"}\mathit{name}\text{"}$$

- modules doivent être instanciés par l'*embedder* (JS, OS)
- on doit fournir des définitions pour tous les *imports*
- on peut alors invoquer les fonctions exportées

Fonctions

(fonctions) une séquence de valeur en entrée et en sortie, type de retour, définie localement ou importée :

$$\begin{aligned} \mathrm{f} ::= & \; ex^* \; \texttt{func} \; \mathrm{tf} \; \texttt{local} \; \mathrm{t}^* \; e^* \\ | & \; ex^* \; \texttt{func} \; \mathrm{tf} \; im \end{aligned}$$

Fonctions

(fonctions) une séquence de valeur en entrée et en sortie, type de retour, définie localement ou importée :

$$\begin{aligned} \mathrm{f} ::= & \; ex^* \; \texttt{func} \; \mathrm{tf} \; \texttt{local} \; \mathrm{t}^* \; e^* \\ | & \; ex^* \; \texttt{func} \; \mathrm{tf} \; im \end{aligned}$$

- les fonctions peuvent s'appeler mutuellement et récursivement

Fonctions

(fonctions) une séquence de valeur en entrée et en sortie, type de retour, définie localement ou importée :

$$\begin{aligned} \mathrm{f} ::= & \; ex^* \; \texttt{func} \; \mathrm{tf} \; \texttt{local} \; \mathrm{t}^* \; e^* \\ | & \; ex^* \; \texttt{func} \; \mathrm{tf} \; im \end{aligned}$$

- les fonctions peuvent s'appeler mutuellement et récursivement
- seulement de seconde classe

Fonctions

(fonctions) une séquence de valeur en entrée et en sortie, type de retour, définie localement ou importée :

$$\begin{aligned} \mathrm{f} ::= & \; ex^* \; \texttt{func} \; \mathrm{tf} \; \texttt{local} \; \mathrm{t}^* \; e^* \\ | & \; ex^* \; \texttt{func} \; \mathrm{tf} \; im \end{aligned}$$

- les fonctions peuvent s'appeler mutuellement et récursivement
- seulement de seconde classe
- pile d'appel pas exposée, inatteignable par un programme bogué ou malveillant

Instructions

- calculs basés sur une machine à pile

Instructions

- calculs basés sur une machine à pile
- code de fonction : séquence d'instructions manipulant des valeurs sur une pile d'opérandes implicite

Instructions

- calculs basés sur une machine à pile
- code de fonction : séquence d'instructions manipulant des valeurs sur une pile d'opérandes implicite
- les arguments sont dépilés et les résultats empilés

Instructions

- calculs basés sur une machine à pile
- code de fonction : séquence d'instructions manipulant des valeurs sur une pile d'opérandes implicite
- les arguments sont dépilés et les résultats empilés
- système de type permet de déterminer statiquement la forme de la pile à n'importe quel moment

Instructions

- calculs basés sur une machine à pile
- code de fonction : séquence d'instructions manipulant des valeurs sur une pile d'opérandes implicite
- les arguments sont dépilés et les résultats empilés
- système de type permet de déterminer statiquement la forme de la pile à n'importe quel moment
- on peut compiler le flot de données entre les instructions sans matérialiser la pile d'opérandes

Instructions

- calculs basés sur une machine à pile
- code de fonction : séquence d'instructions manipulant des valeurs sur une pile d'opérandes implicite
- les arguments sont dépilés et les résultats empilés
- système de type permet de déterminer statiquement la forme de la pile à n'importe quel moment
- on peut compiler le flot de données entre les instructions sans matérialiser la pile d'opérandes
- permet une représentation plus compacte qu'une machine à registres

Instructions

$$e ::= \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end}$$

Instructions

$$
\begin{aligned}
e ::= {} & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i
\end{aligned}
$$

Instructions

$$
\begin{aligned}
e ::= \; & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i \\
& \mid \texttt{br\_table}\ i^+ \mid \texttt{return} \mid \texttt{call}\ i \mid \texttt{call\_indirect}\ \mathrm{tf}
\end{aligned}
$$

Instructions

$$
\begin{aligned}
e ::= \; & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i \\
& \mid \texttt{br\_table}\ i^+ \mid \texttt{return} \mid \texttt{call}\ i \mid \texttt{call\_indirect}\ \mathrm{tf} \\
& \mid \texttt{get\_local}\ i \mid \texttt{set\_local}\ i \mid \texttt{tee\_local}\ i
\end{aligned}
$$

Instructions

$$
\begin{aligned}
e ::= \; & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i \\
& \mid \texttt{br\_table}\ i^+ \mid \texttt{return} \mid \texttt{call}\ i \mid \texttt{call\_indirect}\ \mathrm{tf} \\
& \mid \texttt{get\_local}\ i \mid \texttt{set\_local}\ i \mid \texttt{tee\_local}\ i \\
& \mid \texttt{get\_global}\ i \mid \texttt{set\_global}\ i \mid \mathrm{t}.\texttt{load}\ (\mathrm{tp\_}sx)^?\ \mathrm{a}\ \mathrm{o}
\end{aligned}
$$

Instructions

$$
\begin{aligned}
e ::= \; & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i \\
& \mid \texttt{br\_table}\ i^+ \mid \texttt{return} \mid \texttt{call}\ i \mid \texttt{call\_indirect}\ \mathrm{tf} \\
& \mid \texttt{get\_local}\ i \mid \texttt{set\_local}\ i \mid \texttt{tee\_local}\ i \\
& \mid \texttt{get\_global}\ i \mid \texttt{set\_global}\ i \mid \mathrm{t}.\texttt{load}\ (\mathrm{tp\_}sx)^?\ \mathrm{a}\ \mathrm{o} \\
& \mid \mathrm{t}.\texttt{store}\ \mathrm{tp}^?\ \mathrm{a}\ \mathrm{o} \mid \texttt{current\_memory} \mid \texttt{grow\_memory}
\end{aligned}
$$

Instructions

$$
\begin{aligned}
e ::= {} & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i \\
& \mid \texttt{br\_table}\ i^+ \mid \texttt{return} \mid \texttt{call}\ i \mid \texttt{call\_indirect}\ \mathrm{tf} \\
& \mid \texttt{get\_local}\ i \mid \texttt{set\_local}\ i \mid \texttt{tee\_local}\ i \\
& \mid \texttt{get\_global}\ i \mid \texttt{set\_global}\ i \mid \mathrm{t}.\texttt{load}\ (\mathrm{tp\_}sx)^?\ \mathrm{a}\ \mathrm{o} \\
& \mid \mathrm{t}.\texttt{store}\ \mathrm{tp}^?\ \mathrm{a}\ \mathrm{o} \mid \texttt{current\_memory} \mid \texttt{grow\_memory} \\
& \mid \mathrm{t}.\texttt{const}\ \mathrm{c} \mid \mathrm{t}.\mathit{unop}_\mathrm{t} \mid \mathrm{t}.\mathit{binop}_\mathrm{t} \mid \mathrm{t}.\mathit{testop}_\mathrm{t} \mid \mathrm{t}.\mathit{relop}_\mathrm{t}
\end{aligned}
$$

Instructions

$$
\begin{aligned}
e ::= {} & \texttt{unreachable} \mid \texttt{nop} \mid \texttt{drop} \mid \texttt{select} \mid \texttt{block}\ \mathrm{tf}\ e^*\ \texttt{end} \\
& \mid \texttt{loop}\ \mathrm{tf}\ e^*\ \texttt{end} \mid \texttt{if}\ \mathrm{tf}\ e^*\ \texttt{else}\ e^*\ \texttt{end} \mid \texttt{br}\ i \mid \texttt{br\_if}\ i \\
& \mid \texttt{br\_table}\ i^+ \mid \texttt{return} \mid \texttt{call}\ i \mid \texttt{call\_indirect}\ \mathrm{tf} \\
& \mid \texttt{get\_local}\ i \mid \texttt{set\_local}\ i \mid \texttt{tee\_local}\ i \\
& \mid \texttt{get\_global}\ i \mid \texttt{set\_global}\ i \mid \mathrm{t}.\texttt{load}\ (\mathrm{tp\_sx})^?\ \mathrm{a}\ \mathrm{o} \\
& \mid \mathrm{t}.\texttt{store}\ \mathrm{tp}^?\ \mathrm{a}\ \mathrm{o} \mid \texttt{current\_memory} \mid \texttt{grow\_memory} \\
& \mid \mathrm{t}.\texttt{const}\ \mathrm{c} \mid \mathrm{t}.\mathit{unop}_\mathrm{t} \mid \mathrm{t}.\mathit{binop}_\mathrm{t} \mid \mathrm{t}.\mathit{testop}_\mathrm{t} \mid \mathrm{t}.\mathit{relop}_\mathrm{t} \\
& \mid \mathrm{t}.\mathit{cvtop}\ \mathrm{t}_s\mathrm{x}^?
\end{aligned}
$$

Traps

- instructions peuvent produire un *trap*

Traps

- instructions peuvent produire un *trap*
- interruption immédiate de l'exécution

Traps

- instructions peuvent produire un *trap*
- interruption immédiate de l'exécution
- impossible à gérer en WebAssembly (pour le moment)

Traps

- instructions peuvent produire un *trap*
- interruption immédiate de l'exécution
- impossible à gérer en WebAssembly (pour le moment)
- géré par l'*embedder*

Traps

- instructions peuvent produire un *trap*
- interruption immédiate de l'exécution
- impossible à gérer en WebAssembly (pour le moment)
- géré par l'*embedder*
- en JS : lève une exception contenant une *stacktrace* avec les *stack frames* JS et wasm

Types

(value types)

$$\mathrm{t} ::= \texttt{i32} \mid \texttt{i64} \mid \texttt{f32} \mid \texttt{f64}$$

Types

(value types)

$$\mathrm{t} ::= \texttt{i32} \mid \texttt{i64} \mid \texttt{f32} \mid \texttt{f64}$$

(packed types)

$$\mathrm{tp} ::= \texttt{i8} \mid \texttt{i16} \mid \texttt{i32}$$

Types

(value types)

$$\mathrm{t} ::= \texttt{i32} \mid \texttt{i64} \mid \texttt{f32} \mid \texttt{f64}$$

(packed types)

$$\mathrm{tp} ::= \texttt{i8} \mid \texttt{i16} \mid \texttt{i32}$$

(function types)

$$\mathrm{tf} ::= \mathrm{t}^* \to \mathrm{t}^*$$

Types

(value types)

$$\mathrm{t} ::= \texttt{i32} \mid \texttt{i64} \mid \texttt{f32} \mid \texttt{f64}$$

(packed types)

$$\mathrm{tp} ::= \texttt{i8} \mid \texttt{i16} \mid \texttt{i32}$$

(function types)

$$\mathrm{tf} ::= \mathrm{t}^* \to \mathrm{t}^*$$

(global types)

$$\mathrm{tg} ::= \texttt{mut}^? \ \mathrm{t}$$

Types

- entiers et flottants IEEE 754, chacun en 32 et 64 bits

Types

- entiers et flottants IEEE 754, chacun en 32 et 64 bits
- `i32` pour adresses mémoire linéaire et index tables de fonction

Types

- entiers et flottants IEEE 754, chacun en 32 et 64 bits
- `i32` pour adresses mémoire linéaire et index tables de fonction
- conversion (et réinterprétation) possible entre les 4 types (si de même taille)

Types

- entiers et flottants IEEE 754, chacun en 32 et 64 bits
- `i32` pour adresses mémoire linéaire et index tables de fonction
- conversion (et réinterprétation) possible entre les 4 types (si de même taille)
- pas de distinction entre entiers signés et non-signés

Types

- entiers et flottants IEEE 754, chacun en 32 et 64 bits
- `i32` pour adresses mémoire linéaire et index tables de fonction
- conversion (et réinterprétation) possible entre les 4 types (si de même taille)
- pas de distinction entre entiers signés et non-signés
- suffixe d'extension de signe `_u` ou `_s` si besoin

Opérateurs

$$sx ::= s \mid u$$

Opérateurs

$$sx ::= \texttt{s} \mid \texttt{u}$$

$$cvtop ::= \texttt{convert} \mid \texttt{reinterpret}$$

Opérateurs

$$\begin{aligned} sx &::= \texttt{s} \mid \texttt{u} \\ cvtop &::= \texttt{convert} \mid \texttt{reinterpret} \\ unop_{iN} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \end{aligned}$$

Opérateurs

$$
\begin{aligned}
\mathit{sx} &::= \texttt{s} \mid \texttt{u} \\
\mathit{cvtop} &::= \texttt{convert} \mid \texttt{reinterpret} \\
\mathit{unop}_{\mathrm{iN}} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \\
\mathit{unop}_{\mathrm{fN}} &::= \texttt{neg} \mid \texttt{abs} \mid \texttt{ceil} \mid \texttt{floor} \\
&\quad \mid \texttt{trunc} \mid \texttt{nearest} \mid \texttt{sqrt}
\end{aligned}
$$

Opérateurs

$$
\begin{aligned}
sx &::= \texttt{s} \mid \texttt{u} \\
cvtop &::= \texttt{convert} \mid \texttt{reinterpret} \\
unop_{iN} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \\
unop_{fN} &::= \texttt{neg} \mid \texttt{abs} \mid \texttt{ceil} \mid \texttt{floor} \\
&\quad \mid \texttt{trunc} \mid \texttt{nearest} \mid \texttt{sqrt} \\
binop_{iN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div}_{sx} \mid \texttt{rem}_{sx} \mid \texttt{and} \mid \texttt{or} \\
&\quad \mid \texttt{xor} \mid \texttt{shl} \mid \texttt{shr}_{sx} \mid \texttt{rotl} \mid \texttt{rotr}
\end{aligned}
$$

Opérateurs

$$
\begin{aligned}
sx &::= \texttt{s} \mid \texttt{u} \\
cvtop &::= \texttt{convert} \mid \texttt{reinterpret} \\
unop_{iN} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \\
unop_{fN} &::= \texttt{neg} \mid \texttt{abs} \mid \texttt{ceil} \mid \texttt{floor} \\
&\quad \mid \texttt{trunc} \mid \texttt{nearest} \mid \texttt{sqrt} \\
binop_{iN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div}_{sx} \mid \texttt{rem}_{sx} \mid \texttt{and} \mid \texttt{or} \\
&\quad \mid \texttt{xor} \mid \texttt{shl} \mid \texttt{shr}_{sx} \mid \texttt{rotl} \mid \texttt{rotr} \\
binop_{fN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div} \mid \texttt{min} \mid \texttt{max} \mid \texttt{copysign}
\end{aligned}
$$

Opérateurs

$$
\begin{aligned}
\mathit{sx} &::= \texttt{s} \mid \texttt{u} \\
\mathit{cvtop} &::= \texttt{convert} \mid \texttt{reinterpret} \\
\mathit{unop}_{\mathrm{iN}} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \\
\mathit{unop}_{\mathrm{fN}} &::= \texttt{neg} \mid \texttt{abs} \mid \texttt{ceil} \mid \texttt{floor} \\
&\quad \mid \texttt{trunc} \mid \texttt{nearest} \mid \texttt{sqrt} \\
\mathit{binop}_{\mathrm{iN}} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div}_{\mathit{sx}} \mid \texttt{rem}_{\mathit{sx}} \mid \texttt{and} \mid \texttt{or} \\
&\quad \mid \texttt{xor} \mid \texttt{shl} \mid \texttt{shr}_{\mathit{sx}} \mid \texttt{rotl} \mid \texttt{rotr} \\
\mathit{binop}_{\mathrm{fN}} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div} \mid \texttt{min} \mid \texttt{max} \mid \texttt{copysign} \\
\mathit{testop}_{\mathrm{iN}} &::= \texttt{eqz}
\end{aligned}
$$

Opérateurs

$$
\begin{aligned}
sx &::= \texttt{s} \mid \texttt{u} \\
cvtop &::= \texttt{convert} \mid \texttt{reinterpret} \\
unop_{iN} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \\
unop_{fN} &::= \texttt{neg} \mid \texttt{abs} \mid \texttt{ceil} \mid \texttt{floor} \\
&\quad \mid \texttt{trunc} \mid \texttt{nearest} \mid \texttt{sqrt} \\
binop_{iN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div}_{sx} \mid \texttt{rem}_{sx} \mid \texttt{and} \mid \texttt{or} \\
&\quad \mid \texttt{xor} \mid \texttt{shl} \mid \texttt{shr}_{sx} \mid \texttt{rotl} \mid \texttt{rotr} \\
binop_{fN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div} \mid \texttt{min} \mid \texttt{max} \mid \texttt{copysign} \\
testop_{iN} &::= \texttt{eqz} \\
relop_{iN} &::= \texttt{eq} \mid \texttt{ne} \mid \texttt{lt}_{sx} \mid \texttt{gt}_{sx} \mid \texttt{le}_{sx} \mid \texttt{ge}_{sx}
\end{aligned}
$$

Opérateurs

$$
\begin{aligned}
sx &::= \texttt{s} \mid \texttt{u} \\
cvtop &::= \texttt{convert} \mid \texttt{reinterpret} \\
unop_{iN} &::= \texttt{clz} \mid \texttt{ctz} \mid \texttt{popcnt} \\
unop_{fN} &::= \texttt{neg} \mid \texttt{abs} \mid \texttt{ceil} \mid \texttt{floor} \\
&\quad \mid \texttt{trunc} \mid \texttt{nearest} \mid \texttt{sqrt} \\
binop_{iN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div}_{sx} \mid \texttt{rem}_{sx} \mid \texttt{and} \mid \texttt{or} \\
&\quad \mid \texttt{xor} \mid \texttt{shl} \mid \texttt{shr}_{sx} \mid \texttt{rotl} \mid \texttt{rotr} \\
binop_{fN} &::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div} \mid \texttt{min} \mid \texttt{max} \mid \texttt{copysign} \\
testop_{iN} &::= \texttt{eqz} \\
relop_{iN} &::= \texttt{eq} \mid \texttt{ne} \mid \texttt{lt}_{sx} \mid \texttt{gt}_{sx} \mid \texttt{le}_{sx} \mid \texttt{ge}_{sx} \\
relop_{fN} &::= \texttt{eq} \mid \texttt{ne} \mid \texttt{lt} \mid \texttt{gt} \mid \texttt{le} \mid \texttt{ge}
\end{aligned}
$$

Variables locales

- une fonction déclare des variables locales *muables* en donnant leur type

Variables locales

- une fonction déclare des variables locales *muables* en donnant leur type
- elles sont *zero-initialized*

Variables locales

- une fonction déclare des variables locales *muables* en donnant leur type
- elles sont *zero-initialized*
- lues et écrites par indice avec `get_local`, `set_local` et `tee_local` (pile inchangée)

Variables locales

- une fonction déclare des variables locales *muables* en donnant leur type
- elles sont *zero-initialized*
- lues et écrites par indice avec `get_local`, `set_local` et `tee_local` (pile inchangée)
- les indices commencent avec les paramètres de la fonction (donc muables également)

Variables globales

$$\mathrm{glob} ::= \mathit{ex}^* \texttt{global}\ \mathrm{tg}\ e^* \mid \mathit{ex}^* \texttt{global}\ \mathrm{tg}\ \mathrm{im}$$

Variables globales

$$\mathrm{glob} ::= \mathit{ex}^* \texttt{global}\ \mathrm{tg}\ e^* \mid \mathit{ex}^* \texttt{global}\ \mathrm{tg}\ \mathrm{im}$$

Variables globales

$$\mathtt{glob} ::= \mathit{ex}^* \ \texttt{global} \ \mathtt{tg} \ e^* \mid \mathit{ex}^* \ \texttt{global} \ \mathtt{tg} \ \mathtt{im}$$

- un module déclare des variables globales *muables* ou non, typées

Variables globales

$$\mathrm{glob} ::= ex^* \texttt{ global } \mathrm{tg}\ e^* \mid ex^* \texttt{ global } \mathrm{tg}\ \mathrm{im}$$

- un module déclare des variables globales *muables* ou non, typées
- initialisées par une expression constante sans accès à une fonction, table, mémoire ou globale muable

Variables globales

$$\mathrm{glob} ::= \mathit{ex}^* \texttt{global}\ \mathrm{tg}\ e^* \mid \mathit{ex}^* \texttt{global}\ \mathrm{tg}\ \mathrm{im}$$

- un module déclare des variables globales *muables* ou non, typées
- initialisées par une expression constante sans accès à une fonction, table, mémoire ou globale muable
- lues et écrites par indice avec `get_global` et `set_global`

Variables globales

$$\mathrm{glob} ::= ex^* \texttt{ global } \mathrm{tg}\ e^* \mid ex^* \texttt{ global } \mathrm{tg}\ \mathrm{im}$$

- un module déclare des variables globales *muables* ou non, typées
- initialisées par une expression constante sans accès à une fonction, table, mémoire ou globale muable
- lues et écrites par indice avec `get_global` et `set_global`
- peuvent servir à la configuration (e.g. *linking*)

Mémoire

$$\mathit{mem} ::= \mathit{ex}^{*}\ \texttt{memory}\ \mathit{n} \mid \mathit{ex}^{*}\ \texttt{memory}\ \mathit{n}\ \mathit{im}$$

Mémoire

$$\mathit{mem} ::= \mathit{ex}^* \texttt{memory}\ n \mid \mathit{ex}^* \texttt{memory}\ n\ \mathit{im}$$

- la mémoire est un grand tableau d'octets, appelée *mémoire linéaire*

Mémoire

$$\mathit{mem} ::= \mathit{ex}^* \texttt{memory}\ n \mid \mathit{ex}^* \texttt{memory}\ n\ \mathit{im}$$

- la mémoire est un grand tableau d'octets, appelée *mémoire linéaire*
- au plus une *mémoire* par module, partageable via import/export

Mémoire

$$\mathit{mem} ::= \mathit{ex}^* \texttt{memory}\ n \mid \mathit{ex}^* \texttt{memory}\ n\ \mathit{im}$$

- la mémoire est un grand tableau d'octets, appelée *mémoire linéaire*
- au plus une *mémoire* par module, partageable via import/export
- taille initiale peut être augmentée via `grow_memory`

Mémoire

$$\mathit{mem} ::= \mathit{ex}^* \texttt{memory}\ n \mid \mathit{ex}^* \texttt{memory}\ n\ \mathit{im}$$

- la mémoire est un grand tableau d'octets, appelée *mémoire linéaire*
- au plus une *mémoire* par module, partageable via import/export
- taille initiale peut être augmentée via `grow_memory`
- retourne -1 en cas d'échec, à gérer par le programme

Mémoire

$$\mathit{mem} ::= \mathit{ex}^* \texttt{memory}\ n \mid \mathit{ex}^* \texttt{memory}\ n\ \mathit{im}$$

- la mémoire est un grand tableau d'octets, appelée *mémoire linéaire*
- au plus une *mémoire* par module, partageable via import/export
- taille initiale peut être augmentée via `grow_memory`
- retourne -1 en cas d'échec, à gérer par le programme
- taille actuelle obtenue par `current_memory`

Mémoire

$$\mathit{mem} ::= \mathit{ex}^* \texttt{memory}\ n \mid \mathit{ex}^* \texttt{memory}\ n\ \mathit{im}$$

- la mémoire est un grand tableau d'octets, appelée *mémoire linéaire*
- au plus une *mémoire* par module, partageable via import/export
- taille initiale peut être augmentée via `grow_memory`
- retourne -1 en cas d'échec, à gérer par le programme
- taille actuelle obtenue par `current_memory`
- taille et augmentation en *pages* de taille fixée à 64KiB

Accès mémoire

- via les instructions `load` et `store`

Accès mémoire

- via les instructions `load` et `store`
- les adresses sont des entiers non-signés démarrant à 0

Accès mémoire

- via les instructions `load` et `store`
- les adresses sont des entiers non-signés démarrant à 0
- accès sont vérifiés dynamiquement, accès hors des bornes produisent un `trap`

Accès mémoire

- via les instructions `load` et `store`
- les adresses sont des entiers non-signés démarrant à 0
- accès sont vérifiés dynamiquement, accès hors des bornes produisent un `trap`
- on peut utiliser les types tp d'entiers *packed* pour charger des valeurs plus petites que `i32`

Accès mémoire

- via les instructions `load` et `store`
- les adresses sont des entiers non-signés démarrant à 0
- accès sont vérifiés dynamiquement, accès hors des bornes produisent un `trap`
- on peut utiliser les types `tp` d'entiers *packed* pour charger des valeurs plus petites que `i32`
- il est possible d'effectuer des accès non-alignés

Boutisme

- mémoire petit-boutiste

Boutisme

- mémoire petit-boutiste
- parce que le matériel actuel converge vers le petit-boutisme

Boutisme

- mémoire petit-boutiste
- parce que le matériel actuel converge vers le petit-boutisme
- plate-formes grand-boutiste doivent faire des conversions explicites

Boutisme

- mémoire petit-boutiste
- parce que le matériel actuel converge vers le petit-boutisme
- plate-formes grand-boutiste doivent faire des conversions explicites
- mais généralement optimisé par les compilateurs

Boutisme

- mémoire petit-boutiste
- parce que le matériel actuel converge vers le petit-boutisme
- plate-formes grand-boutiste doivent faire des conversions explicites
- mais généralement optimisé par les compilateurs
- permet d'avoir une sémantique déterministe et portable

Sécurité mémoire

- mémoire linéaire disjointe du code, de la pile d'exécution etc

Sécurité mémoire

- mémoire linéaire disjointe du code, de la pile d'exécution etc
- les programmes ne peuvent corrompre leur environnement d'exécution, effectuer de *jumps* à des adresses arbitraires ou autres comportements non définis

Sécurité mémoire

- mémoire linéaire disjointe du code, de la pile d'exécution etc
- les programmes ne peuvent corrompre leur environnement d'exécution, effectuer de *jumps* à des adresses arbitraires ou autres comportements non définis
- au pire, un programme bogué ou malveillant peut changer sa propre mémoire

Sécurité mémoire

- mémoire linéaire disjointe du code, de la pile d'exécution etc
- les programmes ne peuvent corrompre leur environnement d'exécution, effectuer de *jumps* à des adresses arbitraires ou autres comportements non définis
- au pire, un programme bogué ou malveillant peut changer sa propre mémoire
- permet d'utiliser d'autres modules wasm sans avoir confiance en eux

Sécurité mémoire

- mémoire linéaire disjointe du code, de la pile d'exécution etc
- les programmes ne peuvent corrompre leur environnement d'exécution, effectuer de *jumps* à des adresses arbitraires ou autres comportements non définis
- au pire, un programme bogué ou malveillant peut changer sa propre mémoire
- permet d'utiliser d'autres modules wasm sans avoir confiance en eux
- permet d'embarquer le moteur wasm sans risque mémoire dans d'autres langages

Flot de contrôle

- pas de *jumps* contrairement aux machine à pile classiques mais un flot de contrôle *structuré*

Flot de contrôle

- pas de *jumps* contrairement aux machine à pile classiques mais un flot de contrôle *structuré*
- garantit par construction que le flot de contrôle a de bonnes propriétés (e.g. pas de branchements vers des blocs avec de mauvaises tailles de piles)

Flot de contrôle

- pas de *jumps* contrairement aux machine à pile classiques mais un flot de contrôle *structuré*
- garantit par construction que le flot de contrôle a de bonnes propriétés (e.g. pas de branchements vers des blocs avec de mauvaises tailles de piles)
- permet au code d'être validé, compilé ou même transformé en forme SSA en une seule passe

Flot de contrôle

- pas de *jumps* contrairement aux machine à pile classiques mais un flot de contrôle *structuré*
- garantit par construction que le flot de contrôle a de bonnes propriétés (e.g. pas de branchements vers des blocs avec de mauvaises tailles de piles)
- permet au code d'être validé, compilé ou même transformé en forme SSA en une seule passe
- permet d'inspecter le code facilement

Structures de contrôle et blocs

- `block`, `loop` et `if` doivent terminer par `end` et être imbriquées correctement pour être considérées bien formées

Structures de contrôle et blocs

- `block`, `loop` et `if` doivent terminer par `end` et être imbriquées correctement pour être considérées bien formées
- la séquence d'instruction $e*$ qui les compose forme un *bloc*

Structures de contrôle et blocs

- `block`, `loop` et `if` doivent terminer par `end` et être imbriquées correctement pour être considérées bien formées
- la séquence d'instruction $e*$ qui les compose forme un *bloc*
- `if` contient deux blocs séparés par un `else` (qu'on peut omettre si bloc vide)

Structures de contrôle et blocs

- `block`, `loop` et `if` doivent terminer par `end` et être imbriquées correctement pour être considérées bien formées
- la séquence d'instruction $e*$ qui les compose forme un *bloc*
- `if` contient deux blocs séparés par un `else` (qu'on peut omettre si bloc vide)
- dépile un opérande `i32` et exécute le premier bloc s'il est différent de 0, l'autre sinon

Structures de contrôle et blocs

- `block`, `loop` et `if` doivent terminer par `end` et être imbriquées correctement pour être considérées bien formées
- la séquence d'instruction $e*$ qui les compose forme un *bloc*
- `if` contient deux blocs séparés par un `else` (qu'on peut omettre si bloc vide)
- dépile un opérande `i32` et exécute le premier bloc s'il est différent de 0, l'autre sinon
- `loop` ne recommence pas au début de la boucle automatiquement…

Branchements

```
let foo = function
  | 0 -> 100
  | 1 -> 200
  | _whatever -> 300
```

Branchements

```
let foo = function
  | 0 -> 100
  | 1 -> 200
  | _whatever -> 300
```

```
(func $foo (param i32) (result i32) (local i32)
  (block (block (block
      local.get 0 ;; 0
      i32.eqz
      br_if 0 ;; -> A
      local.get 0 ;; 1
      i32.const 1
      i32.eq
      br_if 1 ;; -> B
      i32.const 300 ;; _whatever
      local.set 1
      br 2) ;; -> C
    i32.const 100 ;; A
    local.set 1
    br 1) ;; -> C
  i32.const 200 ;; B
  local.set 1)
local.get 1) ;; C
```

Branchements

- les labels référencent les structures de contrôle par leur niveau d'imbrication relatifs

Branchements

- les labels référencent les structures de contrôle par leur niveau d'imbrication relatifs
- il y a une notion de portée : on ne peut référencer que ce dans quoi on est imbriqué

Branchements

- les labels référencent les structures de contrôle par leur niveau d'imbrication relatifs
- il y a une notion de portée : on ne peut référencer que ce dans quoi on est imbriqué
- l'effet d'un branchement dépend de la structure de contrôle que l'on vise :

Branchements

- les labels référencent les structures de contrôle par leur niveau d'imbrication relatifs
- il y a une notion de portée : on ne peut référencer que ce dans quoi on est imbriqué
- l'effet d'un branchement dépend de la structure de contrôle que l'on vise :
    - dans le cas `bloc` ou `if` on fait un saut en avant en reprenant l'exécution après le `end` idoine (comme un `break`)

Branchements

- les labels référencent les structures de contrôle par leur niveau d'imbrication relatifs
- il y a une notion de portée : on ne peut référencer que ce dans quoi on est imbriqué
- l'effet d'un branchement dépend de la structure de contrôle que l'on vise :
  - dans le cas `bloc` ou `if` on fait un saut en avant en reprenant l'exécution après le `end` idoine (comme un `break`)
  - dans le cas `loop` on fait un saut en arrière et la boucle recommence (comme un `continue`)

Fonctions

- le corps d'une fonction est un bloc dont la signature associe la pile vide au résultat de la fonction

Fonctions

- le corps d'une fonction est un bloc dont la signature associe la pile vide au résultat de la fonction
- les arguments sont stockés dans les premières variables locales de la fonction

Fonctions

- le corps d'une fonction est un bloc dont la signature associe la pile vide au résultat de la fonction
- les arguments sont stockés dans les premières variables locales de la fonction
- l'exécution peut se terminer de trois façons différents

Fonctions

- le corps d'une fonction est un bloc dont la signature associe la pile vide au résultat de la fonction
- les arguments sont stockés dans les premières variables locales de la fonction
- l'exécution peut se terminer de trois façons différents
    - en atteignant la fin du bloc (et la pile doit correspondre au type de retour)

Fonctions

- le corps d'une fonction est un bloc dont la signature associe la pile vide au résultat de la fonction
- les arguments sont stockés dans les premières variables locales de la fonction
- l'exécution peut se terminer de trois façons différents
  - en atteignant la fin du bloc (et la pile doit correspondre au type de retour)
  - par un branchement qui cible le bloc de la fonction

Fonctions

- le corps d'une fonction est un bloc dont la signature associe la pile vide au résultat de la fonction
- les arguments sont stockés dans les premières variables locales de la fonction
- l'exécution peut se terminer de trois façons différents
  - en atteignant la fin du bloc (et la pile doit correspondre au type de retour)
  - par un branchement qui cible le bloc de la fonction
  - en utilisant `return` (raccourci pour le point précédent)

Appels directs de fonction

- via l'instruction `call` qui attend l'index de la fonction à appeler

Appels directs de fonction

- via l'instruction `call` qui attend l'index de la fonction à appeler
- les arguments de la fonction sont dépilés au moment de l'appel

Appels directs de fonction

- via l'instruction `call` qui attend l'index de la fonction à appeler
- les arguments de la fonction sont dépilés au moment de l'appel
- les valeurs de retour sont empilées à la fin de l'appel

Appels indirects de fonction

$$\mathit{tab} ::= \mathit{ex}^* \texttt{ table } n\ i^* \mid \mathit{ex}^* \texttt{ table } n\ \mathit{im}$$

Appels indirects de fonction

$$\mathtt{tab} ::= \mathit{ex}^* \ \mathtt{table}\ \mathtt{n}\ \mathtt{i}^* \mid \mathit{ex}^* \ \mathtt{table}\ \mathtt{n}\ \mathtt{im}$$

- via l'instruction `call_indirect` qui attend l'index *dynamique* de la fonction à appeler (parmi la table globale définie par le module)

Appels indirects de fonction

$$\mathtt{tab} ::= \mathit{ex}^* \ \mathtt{table}\ \mathtt{n}\ \mathtt{i}^* \mid \mathit{ex}^* \ \mathtt{table}\ \mathtt{n}\ \mathtt{im}$$

- via l'instruction `call_indirect` qui attend l'index *dynamique* de la fonction à appeler (parmi la table globale définie par le module)
- les fonctions n'ont pas besoin d'avoir le même type

Appels indirects de fonction

$$\mathit{tab} ::= \mathit{ex}^* \texttt{table}\ n\ i^* \mid \mathit{ex}^* \texttt{table}\ n\ \mathit{im}$$

- via l'instruction `call_indirect` qui attend l'index *dynamique* de la fonction à appeler (parmi la table globale définie par le module)
- les fonctions n'ont pas besoin d'avoir le même type
- type attendu fourni au moment de l'appel et vérifié dynamiquement, `trap` en cas d'erreur de type ou accès hors des bornes de la table

Appels indirects de fonction

$$\mathtt{tab} ::= \mathit{ex}^* \ \mathtt{table}\ \mathtt{n}\ \mathtt{i}^* \mid \mathit{ex}^* \ \mathtt{table}\ \mathtt{n}\ \mathtt{im}$$

- via l'instruction `call_indirect` qui attend l'index *dynamique* de la fonction à appeler (parmi la table globale définie par le module)
- les fonctions n'ont pas besoin d'avoir le même type
- type attendu fourni au moment de l'appel et vérifié dynamiquement, `trap` en cas d'erreur de type ou accès hors des bornes de la table
- d'après l'expérience d'*asm.js* : simplifie la représentation des pointeurs sur fonction

Divers

- possibilité d'importer des fonctions (externes) et donc de communiquer avec l'embedder

Divers

- possibilité d'importer des fonctions (externes) et donc de communiquer avec l'embedder
- design évite presque tout non déterminisme (sauf *NaN payloads*, *resource exhaustion*, *host functions*)

Divers

- possibilité d'importer des fonctions (externes) et donc de communiquer avec l'embedder
- design évite presque tout non déterminisme (sauf *NaN payloads*, *resource exhaustion*, *host functions*)
- sémantique opérationnelle à petit pas (une page de règles de réductions)

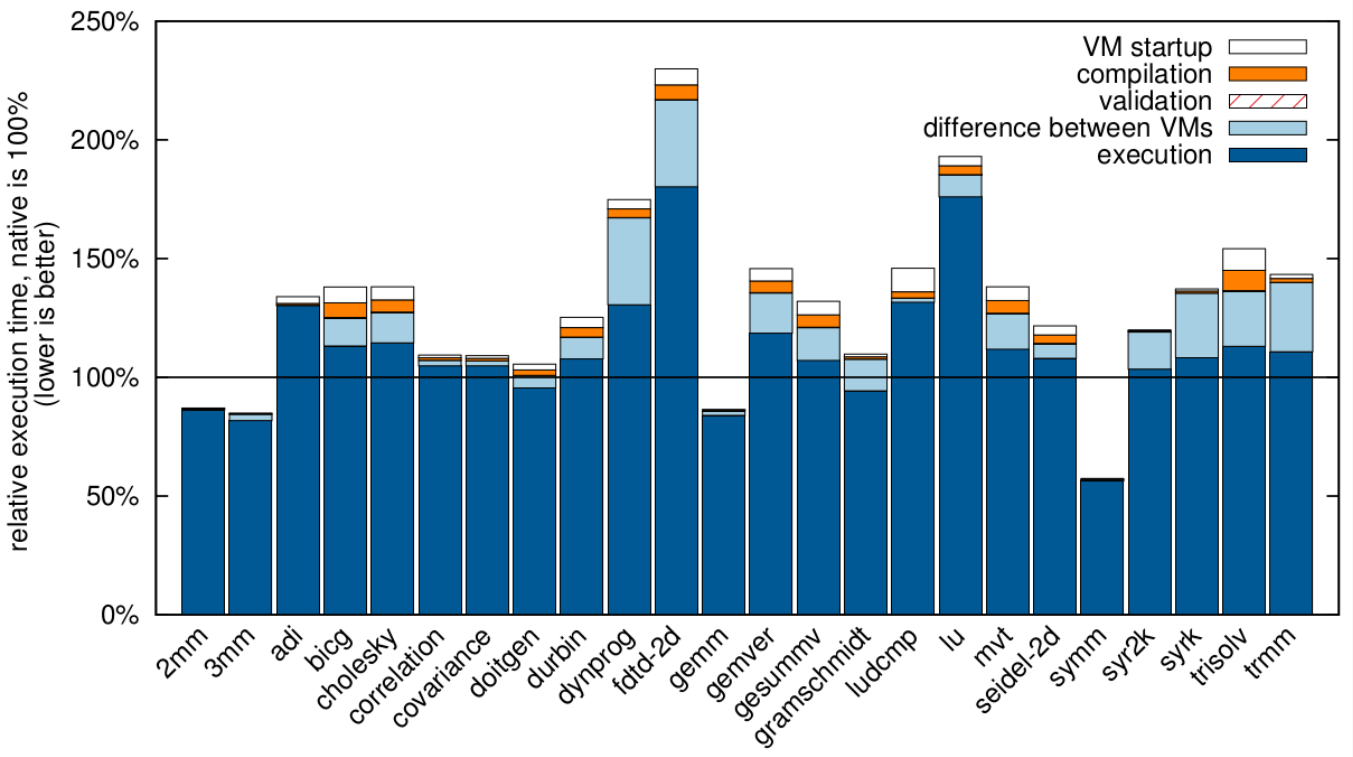
Divers

- possibilité d'importer des fonctions (externes) et donc de communiquer avec l'embedder
- design évite presque tout non déterminisme (sauf *NaN payloads*, *resource exhaustion*, *host functions*)
- sémantique opérationnelle à petit pas (une page de règles de réductions)
- système de type simple formalisé (une demie page de règles de typage, preuve de *soundness* : absence de comportements indéfinis dans la sémantique)

Divers

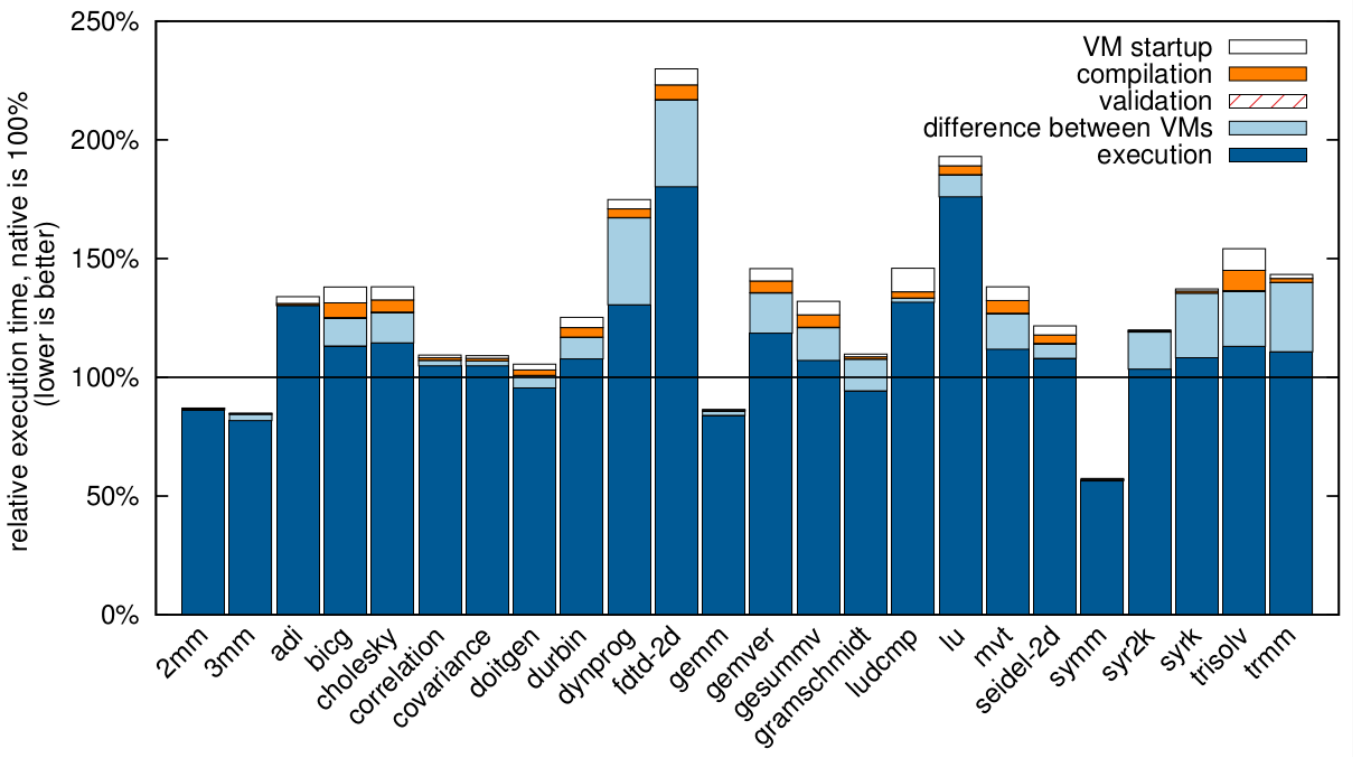
- possibilité d'importer des fonctions (externes) et donc de communiquer avec l'embedder
- design évite presque tout non déterminisme (sauf *NaN payloads*, *resource exhaustion*, *host functions*)
- sémantique opérationnelle à petit pas (une page de règles de réductions)
- système de type simple formalisé (une demie page de règles de typage, preuve de *soundness* : absence de comportements indéfinis dans la sémantique)
- format binaire correspondant à la syntaxe abstraite ; format texte disponible (S-expressions, possibilité de nommer les paramètres et fonctions etc.)

Benchmarks



Relative execution time of the PolyBenchC benchmarks.

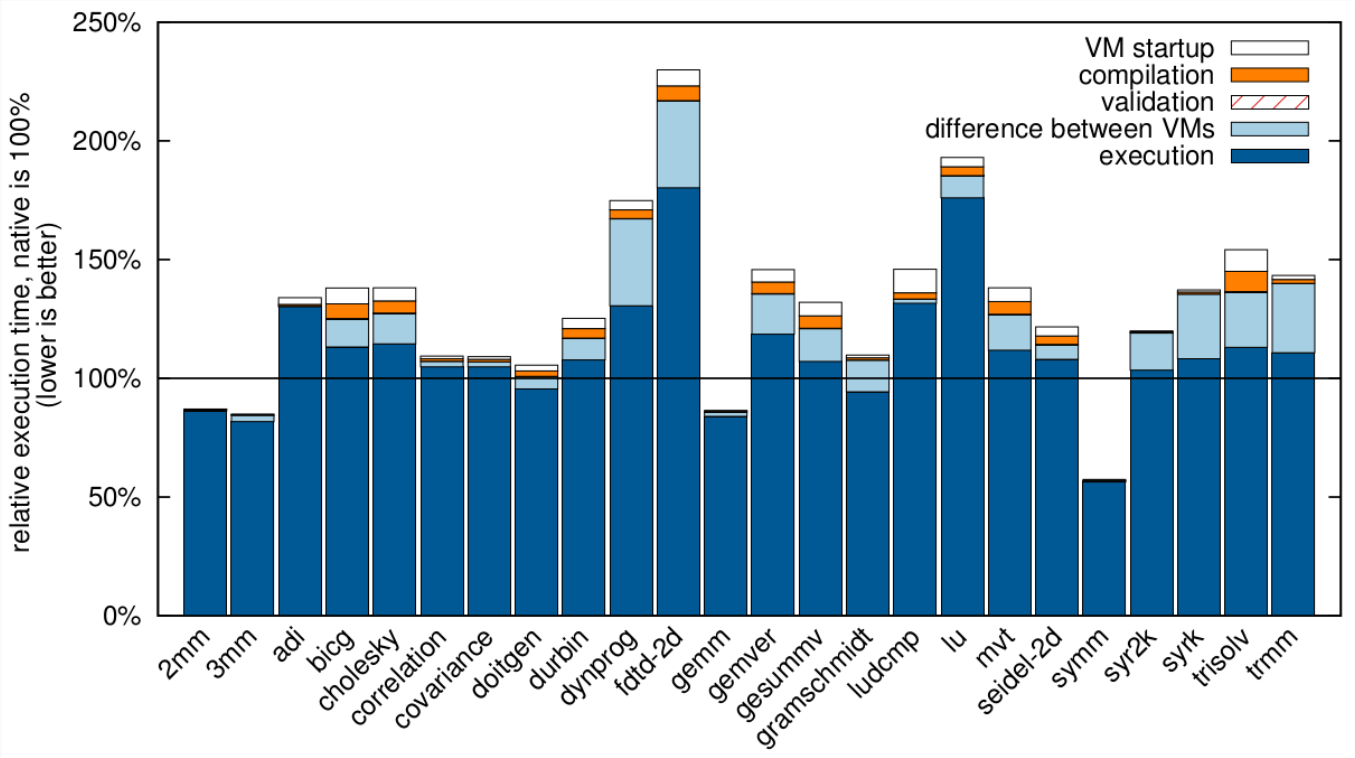
Benchmarks



Relative execution time of the PolyBenchC benchmarks.

7 benchmarks within 10% of native and nearly all of them within 2× of native.

Benchmarks

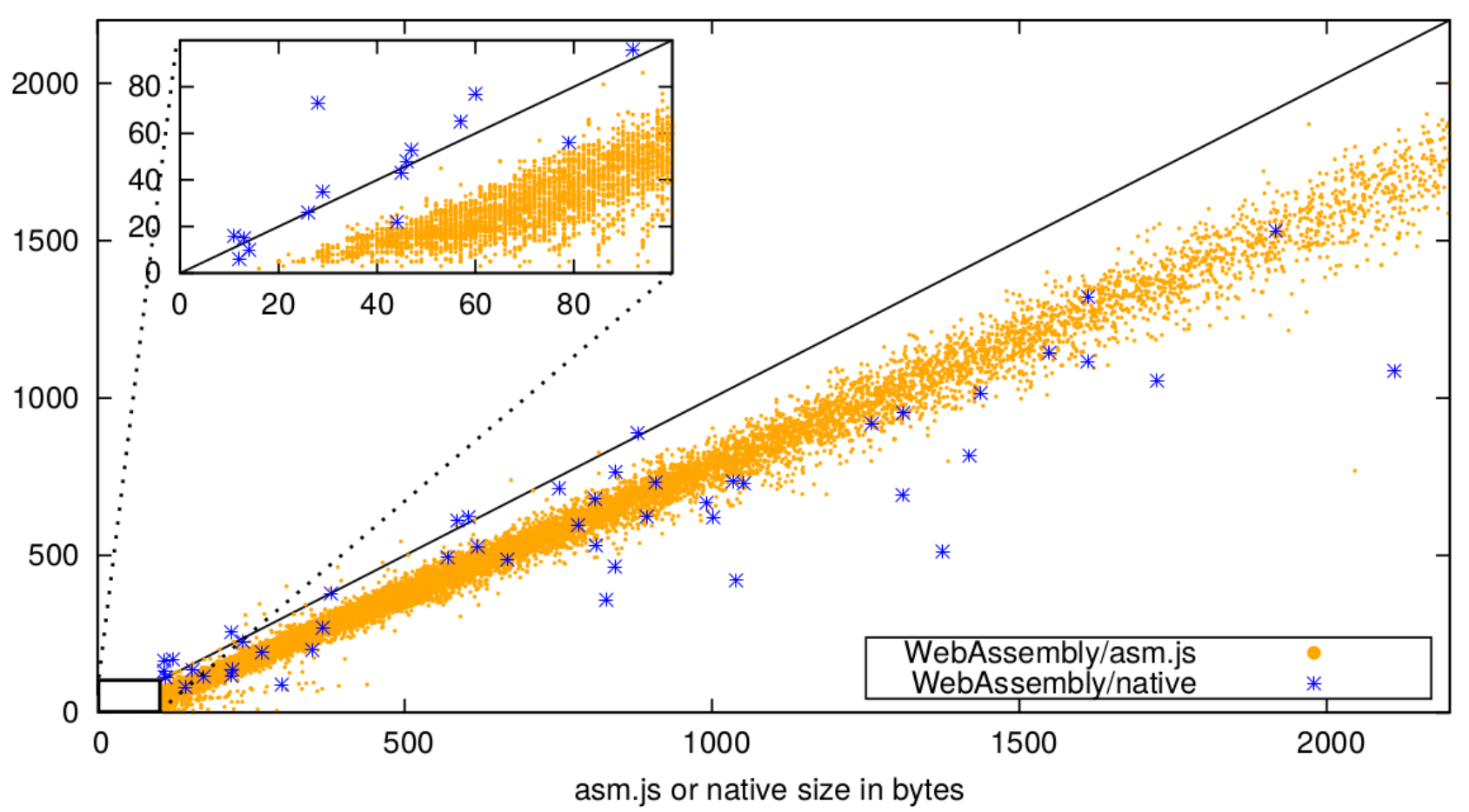


Relative execution time of the PolyBenchC benchmarks.

7 benchmarks within 10% of native and nearly all of them within 2× of native.

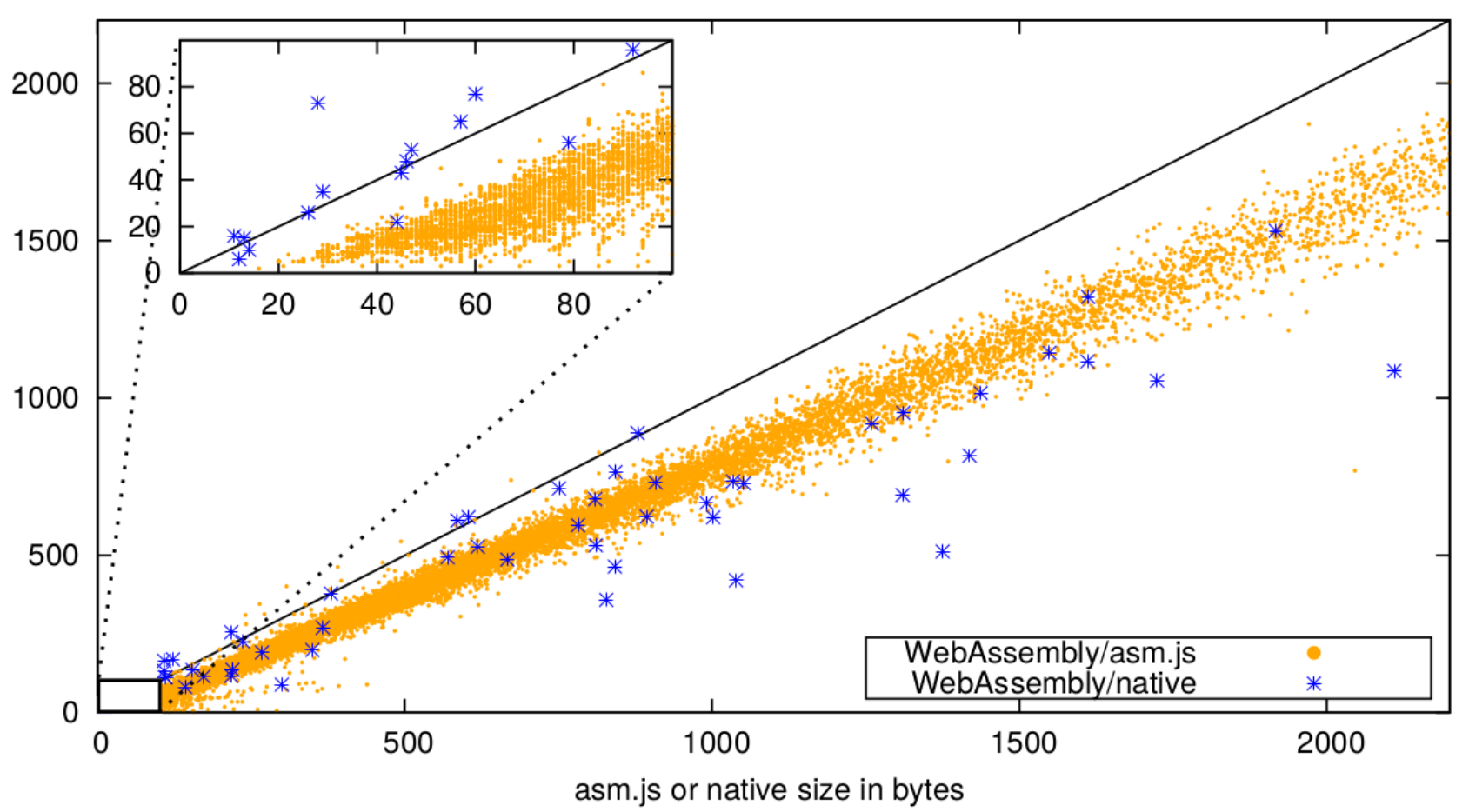
En moyenne wasm est 33.7% plus rapide qu'asm.js. Usage mémoire de la validation entre 1% et 3%.

Benchmarks



Binary size of WebAssembly compared to asm.js and native.

Benchmarks



Binary size of WebAssembly compared to asm.js and native.

En moyenne la taille du code wasm est 62.5% celle d'asm.js et 85.3% celle du x86-64 natif.

Démo

- wasm-demo.zapashcanon.fr

Ajouts

- récents : *bulk memory operations*, *reference types*, *multi-value wasm*, *threads* (*shared memories* et *atomic memory accesses*)

Ajouts

- récents : *bulk memory operations*, *reference types*, *multi-value wasm*, *threads* (*shared memories* et *atomic memory accesses*)
- à venir : *fixed-width simd*, *garbage collector*, *tail-call*, *function references*, *flexible vectors*, *branch hinting* …

Ajouts

- récents : *bulk memory operations*, *reference types*, *multi-value wasm*, *threads* (*shared memories* et *atomic memory accesses*)
- à venir : *fixed-width simd*, *garbage collector*, *tail-call*, *function references*, *flexible vectors*, *branch hinting* …
- WASI (une interface système pour WebAssembly)

Conclusion

- sujet de thèse : design, formalisation et implémentation d'un glaneur de cellules adapté à WebAssembly

Conclusion

- sujet de thèse : design, formalisation et implémentation d'un glaneur de cellules adapté à WebAssembly
- avec Pierre Chambart, Jean-Christophe Filliâtre et Vincent Laviron

Conclusion

- sujet de thèse : design, formalisation et implémentation d'un glaneur de cellules adapté à WebAssembly
- avec Pierre Chambart, Jean-Christophe Filliâtre et Vincent Laviron
- wasm-demo.zapashcanon.fr

Conclusion

- sujet de thèse : design, formalisation et implémentation d'un glaneur de cellules adapté à WebAssembly
- avec Pierre Chambart, Jean-Christophe Filliâtre et Vincent Laviron
- wasm-demo.zapashcanon.fr
- git.zapashcanon.fr/zapashcanon/presentation-wasm

Conclusion

- sujet de thèse : design, formalisation et implémentation d'un glaneur de cellules adapté à WebAssembly
- avec Pierre Chambart, Jean-Christophe Filliâtre et Vincent Laviron
- wasm-demo.zapashcanon.fr
- git.zapashcanon.fr/zapashcanon/presentation-wasm
- fs.zapashcanon.fr/pdf/presentation-wasm.pdf