

2020

DOSSIER DE CANDIDATURE
APPLICATION

Cochez le concours sur lequel vous candidatez
Check the competition exam for which you are applying

- CRCN - (Chargés de recherche de classe normale / Young graduate scientist position)
- DR2 - (Directeurs de recherche de deuxième classe / Senior researcher position)

Nom¹ : BODIN
Last name

Prénom : Martin
First name

Sexe : F M
Sex

Nom utilisé pour vos publications (facultatif) :
Name used for your publications (optional):

¹Il s'agit du nom usuel figurant sur vos pièces d'identité
It is the name appearing on your identity cards

**DEPOT DE VOTRE CANDIDATURE
SUBMITTING YOUR APPLICATION**

Le dossier de candidature doit comprendre :

- Formulaire 1 : Parcours professionnel (page 4)
- Formulaire 2 : Description synthétique de l'activité antérieure (page 6)
- Formulaire 3 : Contributions majeures (page 7)
- Formulaire 4 : Programme de recherche (page 10)
- Formulaire 5 : Liste complète des contributions (page 15)

CRCN :

- Les rapports de thèse ou de doctorat (page 18 pour les rapports de thèse et page 24 pour le rapport de soutenance)
- Une copie des derniers titres et diplômes (page 26)
- Une photographie récente de la candidate / du candidat (facultative, page 17)

DR2 :

- Les rapports d'habilitation à diriger des recherches (si applicable)
- Une copie des derniers titres et diplômes
- Une photographie récente de la candidate / du candidat (facultative)

The application file must include:

- *Form 1: Professional history (page 4)*
- *Form 2: Summary of your past activity (page 6)*
- *Form 3: Major contributions (page 7)*
- *Form 4: Research program (page 10)*
- *Form 5: Complete list of contributions (page 15)*

CRCN:

- *PhD dissertation reports (page 18 for the dissertation reports and page 24 for the defense report)*
- *A copy of most recent titles and diplomas (page 26)*
- *A recent photograph of the applicant (optional, page 17)*

DR2:

- *Habilitation dissertation reports (if applicable)*
- *A copy of most recent titles and diplomas*
- *A recent photograph of the applicant (optional)*

SOMMAIRE / SUMMARY

Formulaire 1 — Parcours professionnel	4
<i>Form 1 — Professional history</i>	<i>4</i>
Formulaire 2 — Description synthétique de l'activité antérieure	6
<i>Form 2 — Summary of your past activity</i>	<i>6</i>
Formulaire 3 — Contributions majeures	7
<i>Form 3 — Major contributions</i>	<i>7</i>
Formulaire 4 — Programme de recherche	10
<i>Form 4 — Research program</i>	<i>10</i>
Formulaire 5 — Liste complète des contributions	15
<i>Form 5 — Complete list of contributions</i>	<i>15</i>

Formulaire 1 — PARCOURS PROFESSIONNEL

Form 1 — PROFESSIONAL HISTORY

1) Parcours Professionnel / *Professional history*

Situation professionnelle actuelle / *Current professional status*

Statut et fonction² / *Position and Status*²: Research associate

Etablissement (ville - pays) / *Institution (city - country)*: Imperial College London (Londres - Royaume-Uni)

Date d'entrée en fonction / *Start*: 11/06/2018

Expériences professionnelles antérieures / *Previous professional experiences*

Déroulez votre parcours professionnel antérieur à Inria, chez Inria, en détachement ou en mise à disposition.

Detail your professional history, before Inria, at Inria, on secondment or leave.

Date début <i>Start</i>	Date fin <i>End</i>	Etablissement <i>Institution</i>	Fonction et statut <i>Position and status</i>
01/09/2009	01/07/213	ENS Lyon	Élève fonctionnaire-stagiaire
01/09/2013	30/09/2016	Université de Rennes 1 / IRISA	Doctorant contractuel
01/04/2017	01/04/2018	Universidad de Chile / CMM	Post-doctorant
11/06/2018	-	Imperial College London	Research associate

Nombre d'années d'exercice des métiers de la recherche après la thèse : 3

Number of years of professional research experience after the PhD: 3

2) Interruptions de carrière / *Career breaks*

Date début <i>Start</i>	Date fin <i>End</i>	Motif de l'interruption / <i>Reason for interruption</i>
----------------------------	------------------------	--

3) Prix et distinctions / *Prizes and awards*

4) Encadrement d'activités de recherche / *Supervision of research activities*

- Encadrement de Tomás Diaz, ingénieur en M2 à l'Universidad de Chile en 2017–2018, co-encadré par Éric Tanter. Ce stage long avait pour but de concevoir une architecture de test pour l'interpréteur CoqR. Le programme final est disponible à l'adresse <https://github.com/TDiazT/CoqR-Tester> et les visualisations associées à l'adresse <https://coqr.dcc.uchile.cl>. Ce stage a débouché sur une publication [BDT18].
- Encadrement de Benjamin Gunton, étudiant en L3 à l'Imperial College en 2019, co-encadré par Philippa Gardner. Ce stage visait à explorer comment les sémantiques squelettiques [Bod+19] peuvent servir à décider du déterminisme d'un langage de programmation. Ce travail était une première étape pour travailler sur des hyperpropriétés de programmes à l'aide des squelettes. Le déterminisme avait été spécifiquement choisi car simple, tout en étant très différent du cadre de l'interprétation abstraite choisi dans l'article original.
- Encadrement de Rao Xiaojia, étudiant en M1 à l'Imperial College en 2020, co-encadré par Philippa Gardner. Ce stage consiste à prouver en Coq des propriétés d'un interpréteur WebAssembly.

5) Responsabilités collectives / *Responsibilities*

- Membre du comité d'évaluation des artéfacts de POPL 2019.

6) Management (le cas échéant) / *Management (if applicable)*

7) Collaborations, mobilité / *Collaborations, mobility*

- Stage de trois mois en 2015, pendant ma thèse, à l'Imperial College avec Philippa Gardner. J'y ai travaillé sur la conception d'un domaine abstrait pour JavaScript compatible avec la règle de contexte (frame rule) de la logique de séparation. Ces travaux ont abouti à terme à une publication [BJS16].
- J'ai fait deux postdocs : un au Chili avec Éric Tanter, et un au Royaume-Uni avec Philippa Gardner. Ces deux postdocs ont respectivement abouti au projet CoqR et aux sémantiques squelettiques (voir fiches 2 et 3).

8) Enseignement / *Teaching*

Nom du cours	année	université	niveau	séances	heqTD
Introduction à Scheme	2012	INSA Rennes	L1	28h CTD, 13h TP	36,7
Introduction à la programmation fonctionnelle	2013	Rennes 1	L1	26h TP	17,3
	2014			26h TP	17,3
Langages formels	2013	ENS Rennes	L3	30h TD	30
	2014			30h TD	30
Conception et vérification formelle de programmes	2014	ENS Rennes	M1	26h TD	26
Introducción a Coq	2017	Universidad de Chile	M2	22h TD, 3h CM	26,5
Models of Computation	2018	Imperial College London	L2	27h CTD	27
Separation Logic	2019	Imperial College London	M1	28h CTD	28

9) Diffusion de l'information scientifique / *Dissemination of scientific knowledge*

10) Éléments divers / *Other relevant information*

- Exposé invité à la conférence Formal Methods for Statistical Software (FMfSS) en 2019.

Formulaire 2 — DESCRIPTION SYNTHÉTIQUE DE L'ACTIVITÉ ANTÉRIEURE

Form 2 — SUMMARY OF YOUR PAST ACTIVITY

Ma recherche s'est concentrée sur l'étude des langages de script, notamment JavaScript et R. Ces langages sont en effet très utilisés en pratique [Zap] et ont des domaines d'application très variés. De plus, ces langages se caractérisent par de nombreux comportements spéciaux, et donc une sémantique complexe. Ces comportements spéciaux tendent à artificiellement produire un résultat même dans les cas où le programme comporte des erreurs. JavaScript et R vont ainsi convertir implicitement des valeurs numériques en des chaînes de caractères si une fonction attend l'un plutôt que l'autre. Ceci donne un faux sentiment de correction, les erreurs de programmation étant moins visibles : les programmes retournent parfois des valeurs fausses qui ont l'air superficiellement correctes. Par exemple, en JavaScript, si une expression de la forme $x + y$ apparaît dans un contexte où x et y sont des tableaux, de nombreuses heuristiques se chargeront de retrouver une valeur scalaire (un nombre ou un chaîne de caractère). Il s'agit probablement d'une erreur de programmation, mais elle pourra passer inaperçue si le type de retour choisi par ses heuristiques correspond au type attendu. De telles erreurs sont très difficiles à détecter, en particulier dans des domaines statistiques ([HAP14] expose un tel exemple).

Ces comportements spéciaux complexifient énormément l'analyse de programmes. Ils sont donc souvent ignorés ou simplifiés en recherche. Malheureusement, ces hypothèses de recherche sur ces fonctionnalités ne correspondent que rarement aux usages pratiques [Ric+11]. Ces comportements spéciaux sont parfois peu connus des développeurs [Bur11], ce qui peut donner lieu à de nombreuses erreurs logicielles.

Ma recherche consiste à résorber ce décalage entre les analyses de programme et les pratiques industrielles en prenant en compte les comportements spéciaux des langages de script. J'ai majoritairement travaillé dans le cadre de l'assistant de preuve Coq [CHP+84], qui permet un haut niveau de confiance. J'ai suivi deux directions de recherche principales :

- La création de modèles précis de langages de script.
- La conception d'analyses génériques pour ces modèles.

1) Création de modèles précis de langages de script

Cette direction correspond aux fiches 1 et 2. Ses publications principales sont JSCert [Bod+14] et CoqR [BDT18]. Dans les deux cas, il s'agit d'une formalisation la plus complète possible du langage considéré, sans aucune simplification. Cette formalisation a pour but de servir de base de confiance à des développements ultérieurs, tels des analyses de programme. La contrainte clef est l'absence de simplification : il est tentant de simplifier la sémantique du langage, ou de l'exprimer d'une manière plus idiomatique en Coq. Cependant, de telles simplifications occulteraient certains comportements spéciaux, ce qui irait à l'encontre du but final du projet. De plus, de telles simplifications éloigneraient le modèle de sa spécification, le rendant plus difficile à évaluer. Enfin, elles freineraient l'adaptation du modèle aux évolutions du langage formalisé.

Du fait de cette absence de simplifications — mais aussi de la complexité de la sémantique des langages de script — de tels modèles sont souvent très complexes. Ils mettent ainsi à l'épreuve la capacité de Coq à manipuler de telles tailles de sémantiques. Du fait de leur taille, ces modèles sont aussi difficiles à évaluer. JSCert et CoqR ont donc mis en place des méthodes d'évaluation complémentaires, notamment en construisant des modèles lisibles (et donc vérifiables), mais aussi en mettant en place des architectures de tests.

2) Conception d'analyses génériques

Les modèles sémantiques JSCert et CoqR ont été conçus dans le but de servir à certifier des analyses sur les langages considérés. De telles analyses ont par exemple déjà été mises en place dans des projets ambitieux comme Verasco [Jou+15] : ce dernier se base sur la formalisation de Clight du projet CompCert [Ler+08] et consiste en un analyseur de Clight certifié en Coq. Cependant, JSCert et CoqR sont des modèles particulièrement grands : JSCert compte plus de 900 règles de dérivation et CoqR plus de 2 000. À titre de comparaison, le langage Clight tel que spécifié dans le projet CompCert [Ler+08] contient environ 200 règles². De tels tailles de sémantique ont nécessité l'introduction d'un formalisme facilitant la définition et la preuve d'analyses : les sémantiques squelettiques (voir fiche 3 ou la publication [Bod+19]).

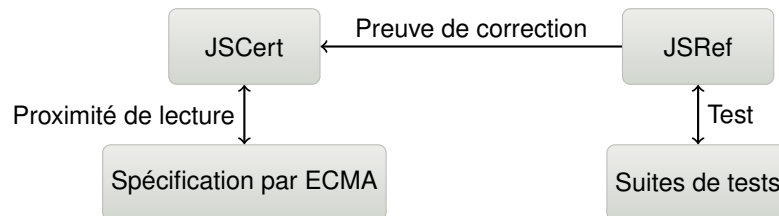
Les sémantiques squelettiques permettent de définir une analyse dans son cas le plus général, pour ensuite l'appliquer à des formalisations effectives du langage de programmation considéré. Ma recherche est cependant restée très théorique jusque là, et aucun analyseur n'a en pratique été défini pour JavaScript ou R. L'application du formalisme des sémantiques squelettiques à des analyses pratiques est une direction importante de ma recherche future. C'est par exemple dans ce but que j'ai conçu un domaine abstrait pour JavaScript [BJS16]. Ce domaine abstrait inclut une règle de contexte (frame rule) issue de la logique de séparation, ce qui dans le cas de JavaScript est un problème difficile [GMS12].

² Le langage Clight tel qu'il est spécifié ne contient en fait qu'une soixantaine de règles de dérivation, mais il est défini dans un style différent de celui de JSCert. Les estimations données ici sont des estimations du nombre de règles obtenues si on spécifiait tous ces langages dans le style de JSCert.

Fiche 1 : JSCert**1. Description de la contribution**

Ma première contribution majeure consiste en le projet JSCert. Ce projet vise à formaliser JavaScript en Coq, dans le but de servir comme base à des développements formels sur ce langage de programmation. En particulier, le but consistait à formaliser complètement le langage, sans simplifications. En effet, la complexité de JavaScript réside dans tous ses cas particuliers, parfois inconnus des développeurs : conversion implicite de type, effets de bords de certaines opérations, les différents modes de l'opérateur `eval`, etc. Le fait que ces cas particuliers soient nombreux a poussé beaucoup de travaux [AGD05; Thi05] à les ignorer ou les simplifier, mais en ce faisant, une différence se crée entre ce qu'analysent ces différents travaux et ce que fait JavaScript. Cette différence est connue pour poser des problèmes de sécurité [MT09; MMT09; MMT10]. JSCert vise à précisément combler cette différence.

JSCert est une formalisation particulièrement grande : plus de 900 règles de dérivations. Il est naturel de douter de la correction de cette formalisation elle-même. Il a donc été nécessaire de mettre en place des liens avec le langage formalisé, au travers de sa spécification par ECMA et de ses suites de tests. JSCert a ainsi été conçu pour être aussi proche que possible de la spécification par ECMA, chaque ligne de spécification correspondant à une règle de dérivation pour JSCert et réciproquement. D'autre part, JSCert est accompagné d'un interpréteur (nommé JSRef) qui a été testé sur les suites de tests.

**2. Contribution personnelle du candidat**

J'étais présent dès le début du projet et ai donc pu contribuer à de nombreuses parties de ce dernier. Les deux parties où j'ai le plus contribué sont l'écriture de l'interpréteur JSRef en Coq, ainsi que sa preuve de correction avec l'aide d'Arthur Charguéraud : j'étais le contributeur principal sur ces deux parties. J'ai aussi participé à la formalisation de nombreuses fonctionnalités de JavaScript dans la spécification JSCert, même si je ne suis pas le contributeur principal sur cette partie. Plusieurs versions de JSCert ont été produites, pour plusieurs versions différentes de la spécification par ECMA, en particulier la version 3 et la version 5. Ces versions étaient malheureusement suffisamment différentes pour qu'une grande partie du travail doive être dupliquée : j'ai donc conçu deux interpréteurs dans ce processus, et j'ai participé à l'identification des points améliorables pour construire la seconde version de l'interpréteur.

Ce travail est une collaboration entre plusieurs équipes à Inria et à l'Imperial College, avec Arthur Charguéraud, Daniele Filaretti, Philippa Gardner, Sergio Maffei, Daiva Naudžiūnienė, Alan Schmitt, et Gareth Smith.

3. Originalité et difficulté

JSCert n'est pas la première sémantique formelle de JavaScript, ni même du langage complet (de nombreux travaux ont été faits sur des sous-langages). C'est cependant la première formalisation reliée à la fois à la spécification par ECMA et aux suites de tests, ce qui la rend particulièrement digne de confiance. C'est de plus la première sémantique de JavaScript exécutable dans un assistant de preuve. En conséquence, JSCert a été le premier projet qui a permis de lier formellement les suites de tests à la spécification, et donc d'évaluer la couverture de la spécification par les suites de tests. JSCert est un projet Coq conséquent et compte parmi les plus grandes formalisations de langages de programmation en Coq.

4. Validation et impact

Le projet JSCert a permis de lier la spécification par ECMA avec les suites de test de JavaScript. Ceci a permis de repérer certaines incohérences : parfois le problème se trouvait dans les suites de tests, mais aussi parfois dans la spécification. De nombreuses discussions avec le comité de spécification ECMA TC39 ont ainsi permis d'informer le comité de ces problèmes, et de guider la spécification des versions futures de JavaScript³.

³ En particulier, le travail de communication d'Alan Schmitt avec le comité de spécification est remarquable. Il a influencé plusieurs choix de spécification du comité, notamment sur la valeur de retour du `while` et sur l'usage de notations monadiques dans la spécification.

Ce lien entre la spécification par ECMA et les suites de tests ont aussi permis d'évaluer la couverture de la suite de test vis-à-vis de la spécification, et ainsi de repérer des parties sous-testées. Des expérimentations dans ces parties spécifiques ont mis en avant des bogues dans SpiderMonkey (le moteur JavaScript de Firefox) et V8 (celui de Chrome). Ces bogues ont été reportés et validés par les différents navigateurs.

JSCert est maintenant reconnu dans la communauté comme la formalisation de référence de JavaScript en Coq, et est utilisé dans des projets de recherches tant proches des équipes participantes (par exemple [Fra+17]) que des équipes indépendantes (par exemple <https://querycert.github.io/> utilise JSCert dans sa partie JavaScript).

5. Diffusion

Le code source du projet et les ressources associées sont disponibles en ligne : <http://jscert.org/>. Le projet JSCert a abouti à de nombreuses publications. Les principales publications auxquelles j'ai participé sont [BS13] relative à l'interpréteur pour la version 3 de la spécification par ECMA, et [Bod+14], la publication principale du projet JSCert.

Ce projet m'a aussi conduit à concevoir un domaine abstrait pour JavaScript [BJS16], dans le but de pouvoir l'appliquer à JSCert.

Fiche 2 : CoqR

1. Description de la contribution

Mon premier post-doctorat portait sur R, un langage de programmation très utilisé en statistiques (avec des domaines d'application allant de la biologie à la finance, en passant par le Big Data). L'objectif était de formaliser le langage de programmation en Coq en suivant l'approche de JSCert, dans le but de certifier des programmes R ou de concevoir des analyseurs certifiés. Le projet CoqR est basé sur les mêmes principes que JSCert : il consiste en un interpréteur Coq (d'un cœur) de R associé à deux liens complémentaires à R. D'une part, chaque ligne de l'interpréteur peut être relié à une ligne du code source de l'interpréteur de référence GNU R. D'autre part, l'interpréteur a été testé sur les suites de tests associées. Ce projet m'a donc permis de valider la méthodologie introduite par JSCert sur un autre langage.

2. Contribution personnelle du candidat

Je suis le contributeur principal de ce projet : j'ai conçu le projet et j'ai produit la presque totalité de la formalisation Coq. Ces travaux sont en collaboration avec Tomás Diaz et Éric Tanter. Tomás Diaz était sous ma supervision dans ce projet, et s'occupait de concevoir une architecture de test en Python.

3. Originalité et difficulté

R est un langage de programmation connu pour être très complexe [Bur11], et très peu d'analyseurs existent pour ce langage. À ma connaissance, CoqR est la première formalisation du langage qui ne cherche pas à le simplifier. En terme de formalisation Coq, CoqR est une formalisation imposante : environ 18 000 lignes de définitions Coq. À titre de comparaison, l'interpréteur de JSCert (qui est déjà considéré comme ayant une taille importante) en contient environ 12 500. Cette taille a posé des problèmes de passage à l'échelle pour certaines définitions : ce projet a poussé Coq à ses limites, ce qui a causé des difficultés supplémentaires.

4. Validation et impact

Le projet étant encore récent, il est difficile d'estimer l'impact réel qu'il a eu. Cependant, un soin particulier a été apporté afin que ce projet soit réutilisable par des personnes hors du projet. J'ai ainsi été contacté à plusieurs reprises par des chercheurs indépendants intéressés par des aspects particuliers du projet, ce qui a parfois abouti à des mises à jour mineures de la formalisation pour s'adapter à leurs besoins. Par exemple [GV19] s'intéressait à l'aspect paresseux des calculs dans R. Il manquait malheureusement certaines fonctions dans CoqR pour pouvoir comparer leur approche : nous avons ajouté les fonctions en question suite à leur publication. Nous sommes encore en contact avec ces auteurs.

5. Diffusion

Le code source du projet et les ressources associées sont disponibles en ligne : <https://github.com/Mbodin/CoqR>. Le projet a abouti à deux publications : [Bod18] qui présente le début du projet, et [BDT18] qui présente la version actuelle du projet, avec une application sur des preuves relativement simples.

Fiche 3 : Sémantiques squelettiques / *Skeletal Semantics*

1. Description de la contribution

Devant les tailles des formalisations JSCert et CoqR, produire un projet d'analyse similaire à Verasco [Jou+15] (un analyseur de C certifié en Coq) pour ces langages semblait être une tâche herculéenne. Cependant, une grande différence avec le C est que les modèles mémoire de JavaScript et de R sont relativement simples : pas d'arithmétique pointeur et pas de modèle mémoire faible ou relâché. La seule difficulté concerne donc la taille de ces sémantiques. La difficulté de JavaScript et R est donc différente de celle du C : ils ont un modèle mémoire assez simple, mais une sémantique pleine de comportements spéciaux.

Les sémantiques squelettiques sont un formalisme pour attaquer spécifiquement ce problème : c'est un langage de description de sémantique simple à partir duquel il est possible de définir plusieurs interprétations et de les montrer correctes. Le point fondamental est que ces interprétations peuvent être définies et montrées correctes une fois pour toute, quelque soit la sémantique squelettique. Il est ainsi possible de définir des interprétations concrète et abstraite, de montrer qu'elles sont cohérentes pour toute sémantique squelettique, puis de les appliquer à une sémantique squelettique de JavaScript ou de R. Ces preuves s'appuient sur des propriétés à prouver sur le modèle mémoire, qui dans le cas de JavaScript et R sont assez simples à montrer.

2. Contribution personnelle du candidat

Les sémantiques squelettiques sont issues d'un projet sur le très long terme : le projet initial consistait seulement à formaliser le style de sémantique utilisé dans JSCert (afin d'en construire des analyses). J'étais alors très impliqué tant dans la formalisation papier (qui est devenue le quatrième chapitre de ma thèse [Bod16]) que la formalisation en Coq (dont j'étais l'unique contributeur).

Le projet s'est ensuite généralisé à un style beaucoup plus général, mené par Alan Schmitt, en collaboration avec Philippa Gardner et Thomas Jensen. Je me suis alors surtout focalisé sur la formalisation Coq du projet (et j'étais l'unique contributeur sur cette partie). J'ai cependant continué à contribuer dans les fondements théoriques, sans être le contributeur principal.

3. Originalité et difficulté

De nombreux autres projets de recherches partent du constat qu'il est difficile de construire des analyses pour JavaScript, et ont donc construit des langages de spécification pour ce dernier [Pol+12; GNS13]. Beaucoup de ces langages sont cependant très spécifiques à JavaScript (ou plus généralement, à leur langage cible). Des langages de spécification génériques existent cependant [Sew+10; Mos05; Chu+15]. Le plus connu est \mathbb{K} [RŞ10]. Ce dernier est cependant un langage à la sémantique complexe et parfois peu spécifiée [LG18].

Les sémantiques squelettiques prennent l'approche opposée en définissant un langage de spécification le plus simple possible. Cette simplicité est un choix conscient, et choisir la forme exacte de sémantique à la fois expressive, adaptée pour des langages complexes, et simple a demandé beaucoup de travail. Cette simplicité permet de définir formellement toutes les interprétations des sémantiques squelettiques et leurs preuves de correction en général, indépendamment du langage de programmation sur lequel il sera appliqué. Un tel résultat est crucial pour espérer pouvoir construire une analyse certifiée de langages de programmation complexes comme JavaScript ou R.

4. Validation et impact

Cette ligne de recherche a de nombreux potentiels dans la création d'analyses certifiées pour des langages de programmation complexes. Son application à des langages effectivement utilisés demande cependant de traduire des formalisations telles JSCert ou CoqR en squelette, ce qui risque de prendre du temps, mais qui est prévu.

Un autre intérêt de cette ligne de recherche est qu'elle donne un cadre formel général non seulement pour définir des analyses, mais pour les composer. Il est ainsi possible de considérer comment des types d'analyses très différentes (par exemple l'interprétation abstraite et la logique de séparation) peuvent être combinées en général : il est connu comment combiner ces analyses dans certains cas particuliers, mais il n'existe à ma connaissance aucun cadre général connu pour le faire.

5. Diffusion

Le code source du projet et les ressources associées sont disponibles en ligne : <http://skeletons.inria.fr/>. La publication principale est [Bod+19]. Les versions précédentes des sémantiques squelettiques (lorsque le but était de formaliser le format de JSCert) ont aussi abouti à des publications dans des conférences nationales [BJS13; BJS14], et internationales [BJS15].

Formulaire 4 — PROGRAMME DE RECHERCHE

Form 4 — RESEARCH PROGRAM

Je souhaite candidater dans l'équipe-projet, ou les équipes-projets suivante(s) : Spades

Je ne souhaite pas choisir d'équipe-projet pour l'instant. En cas d'admissibilité, je serai contacté(e) par la présidente ou le président du jury pour discuter de possibles équipes d'accueil.

I would like to apply for the following project-team(s) : Spades

I prefer not to choose a project-team for the moment. If I am considered eligible, I will be contacted by the chair of the jury to discuss possible host teams.

Intitulé du programme de recherche : Sémantiques squelettiques: Analyses certifiées de programmes réels

Title of research program: Skeletal Semantics: Certified Analyses of Real-world Programs

1. Introduction et Contexte

Les langages de programmation modernes sont de plus en plus complexes, et s'exécutent dans des environnements de plus en plus divers. L'utilisation de langages de script (JavaScript, Python, R, etc.) s'est ainsi imposée dans de nombreux domaines, notamment en statistiques et sur le web, mais aussi dans des **systèmes embarqués** [Wil12; tea13; tea15]. Le cas des systèmes embarqués est assez inattendu puisqu'un tel environnement est typiquement très contraint en ressources (en particulier en temps d'exécution, en mémoire, et en énergie), alors que les langages de script sont justement assez peu prédictifs dans ces domaines.

Les langages de script se caractérisent par une **sémantique complexe** composée de très nombreux comportements spéciaux. Ces comportements spéciaux tendent à artificiellement produire un résultat en cas d'erreur. Par exemple, en JavaScript, si une expression de la forme $x + y$ apparaît dans un contexte où x et y sont des tableaux, de nombreuses fonctions vont être appelées (par exemple `Array.prototype.toString`). Ces fonctions peuvent provoquer toutes sortes d'effets de bords et sont redéfinissables à n'importe quel point du programme. Il est probable que l'expression $x + y$ dans un tel environnement soit une erreur de programmation, mais cette expression va bien retourner un résultat. Ce résultat sera souvent faux, mais pourrait par hasard ressembler au résultat attendu et ainsi ne pas être détecté.

Pris à part, chacun de ces comportements spéciaux ne pose pas de problème fondamental. Le problème est que les langages de script sont en général composés de centaines de tels cas particuliers. Ces comportements spéciaux étant souvent méconnus des développeurs [Bur11], il y a un **besoin d'analyse** pour ces langages de programmation. Mais ces comportements spéciaux complexifient énormément l'analyse de programmes. Il arrive qu'ils soient ignorés ou simplifiés par les analyseurs. Malheureusement, ces hypothèses de recherche correspondent rarement aux usages pratiques [Ric+11]. Il y a donc un écart entre ce que l'on attend des analyseurs et leur garanties réelles.

Pour contrer cela, des modèles plus récents cherchent à couvrir la totalité du langage, y compris ses comportements spéciaux. C'est ainsi l'approche de JSCert [Bod+14] et CoqR [BDT18] (voir fiches 1 et 2), mais aussi de nombreux autres travaux [Mem+16; BR15; Ler+08]. Ces modèles formels de langages se distinguent par leur grande taille. Ainsi JSCert totalise plus de 900 règles de dérivation, et CoqR l'équivalent de plus de 2 000. Cette imposante quantité de règles est directement liée aux comportements spéciaux. Cela a deux conséquences majeures. D'une part, il est plus difficile de faire *confiance* à ces formalisations, ce qui nécessite des méthodologies particulières pour les valider. Ces méthodologies sont tout le cœur de travail derrière JSCert et CoqR, et l'assistant de preuve Coq y est fondamental pour pouvoir transmettre cette confiance aux analyses dérivées. D'autre part, utiliser ces formalisations en pratique demande un effort conséquent, limitant très fortement les analyses possibles à partir de ces modèles.

La construction de telles analyses est donc complexe, mais elles s'appliquent directement aux **systèmes réels** : ces modèles complexes correspondent précisément à des contextes industriels, et non à une simplification de ces derniers. Cela implique des contraintes de recherche très spécifiques. En particulier, les analyses doivent **passer à l'échelle** vis-à-vis de la complexité des langages de programmation, et non plus seulement vis-à-vis de la taille des programmes analysés. Les **sémantiques squelettiques** (voir fiche 3) visent exactement à résoudre ce problème de passage à l'échelle vis-à-vis de la complexité des sémantiques. Ce formalisme propose en effet un cadre *simple* dans lequel exprimer des sémantiques, mais surtout des interprétations de ces dernières, ainsi que leurs preuves de correction. Ces interprétations peuvent ensuite servir comme base pour des analyses de programme. Ce programme de recherche vise à développer ce formalisme, en particulier en l'adaptant aux problématiques des systèmes embarqués :

- En développant les fondements théoriques des sémantiques squelettiques, notamment vis-à-vis des analyses de coûts (en mémoire, énergie, ou temps d'exécution),
- En travaillant à des compilations sous contraintes de sémantiques squelettiques,
- En appliquant le formalisme des sémantiques squelettiques à des cas d'analyse concrets.

2. Fondements théoriques des sémantiques squelettiques

Comme indiqué dans la fiche 3, un choix caractéristique des sémantiques squelettiques est d'être simple. Ceci facilite grandement la construction générale d'interprétations, et donc d'analyses. Cependant, les sémantiques squelettiques n'ont pour l'instant été associées qu'à un unique type d'analyses (une sous-classe de l'interprétation abstraite), et ce de manière assez théorique. Ce premier axe consiste donc à développer ces analyses. Les analyses présentées ci-dessous bénéficieront naturellement d'informations plus précises sur les programmes, et pourront donc parfois se compléter. Les liens entre ces différentes analyses me semblent donc aussi importants que les analyses elle-mêmes.

Coûts en temps et énergie Les systèmes embarqués critiques ont de plus en plus besoin d'estimer les coûts d'exécution des programmes exécutés. Spécifier des coûts concrets (en temps d'exécution ou en énergie) est tout à fait exprimable sous la forme d'une interprétation pour sémantique squelettique, par le biais d'une trace d'exécution. Il est ensuite possible d'abstraire ces coûts concrets pour former des analyses. Cependant, les abstractions sur ces coûts [NMR03; TFW00; Mar+14] sont en général complexes. J'aimerais formaliser des analyses de coût déjà existantes dans le contexte des sémantiques squelettiques. Cette voie me semble intéressante sur le long terme, tant pour appliquer le formalisme des sémantiques squelettiques dans un nouveau contexte que pour offrir de nouveaux outils d'analyse de programmes.

Logique de séparation La logique de séparation [IO01; BCO05] a montré pouvoir raisonner efficacement sur des structures complexes et représente ainsi un objectif important pour la formalisation d'analyses. Cette logique a de plus été récemment généralisée par Iris [Jun+17b], qui est rapidement devenue une référence. Iris est formalisée en Coq, mais sa partie sémantique n'a pas été conçue pour des langages complexes : le langage le plus complexe formalisé dans Iris est à ma connaissance RustBelt [Jun+17a] et ne contient que 30 règles de dérivations (à comparer aux 900 règles de JSCert). Les sémantiques squelettiques pourraient fournir à moyen terme un cadre formel intéressant pour la partie sémantique d'Iris, permettant de passer à l'échelle sur des langages de programmation complexes.

Hyperpropriétés de programme J'ai participé à des travaux préliminaires sur le déterminisme des sémantiques squelettiques. Le déterminisme est une propriété très différente de l'interprétation abstraite : là où l'interprétation abstraite cherche à trouver ce que toutes les branches ont en commun, le déterminisme pousse au contraire à trouver des différences entre branches (notamment dans leurs hypothèses). Malgré ces différences, j'ai trouvé des solutions pour construire une interprétation de sémantiques squelettiques exprimant une version forte du déterminisme. Cela semble indiquer que cette direction de recherche est intéressante, bien qu'orthogonale aux autres analyses. À plus long terme, le déterminisme se généralise aux hyperpropriétés de programmes [CS10], qui visent à comparer différentes exécutions possibles d'un même programme. C'est une classe de propriétés assez large, qui inclut par exemple la non-interférence, l'exécution en temps ou énergie constante, mais aussi certains types de correction fonctionnelle de programme, comme la symétrie ou l'associativité de certaines fonctions.

3. Compilation de sémantiques squelettiques

La compilation s'exprime assez facilement dans le formalisme des sémantiques squelettiques : une interprétation squelettique peut prendre la forme de n'importe quel objet mathématique, et rien ne l'interdit d'être elle-même une représentation de la sémantique dans un autre formalisme, ou même d'être un compilateur pour le langage formalisé. J'ai déjà conçu un prototype d'un tel compilateur vers le langage While exprimé sous la forme d'une interprétation squelettique. Cet axe consiste à explorer les compilations de sémantiques squelettiques vers différents formats et sous différentes contraintes.

Langages intermédiaires De nombreux projets d'analyses [Bes+10; Fra+17] se concentrent sur des langages intermédiaires. Ces langages intermédiaires ont une sémantique beaucoup plus simple que le langage d'origine, ce qui simplifie les analyses. Certains de ces langages intermédiaires sont génériques [Gar+19], et partagent ainsi beaucoup de similarités avec les sémantiques squelettiques. Il semble naturel de construire une interprétation squelettique construisant des compilateurs vers ces langages génériques, puis vérifier la correction de ces compilateurs à l'aide des outils de preuve fournis par les sémantiques squelettiques. Un tel compilateur permettra de comparer voire lier des analyses provenant d'autres travaux, évitant l'isolation de ces différents travaux.

Le formalisme \mathbb{K} [RS10] Ce formalisme a pris beaucoup d'importance grâce à plusieurs formalisations ambitieuses [PSR15; BR15]. Cependant, \mathbb{K} étant lui-même assez complexe [LG18], il est difficile de réutiliser ces sémantiques dans d'autres contextes que ceux prévus par les outils associés. Au contraire, la simplicité des sémantiques squelettiques simplifie leur analyse, mais aussi leur compilation. En particulier, il est envisageable de compiler des sémantiques squelettiques vers \mathbb{K} . Ceci permettrait de faire un lien direct entre les deux formalismes et leurs analyses respectives.

Réciproquement, il me semble intéressant de pouvoir compiler (un sous-ensemble de) \mathbb{K} vers les sémantiques squelettiques. Les sémantiques squelettiques permettant de contrôler précisément leur degré d'abstraction, il me semble fondamental qu'une telle compilation puisse aussi abstraire des parties de la sémantique compilée. Par exemple, la formalisation KJS [PSR15] de JavaScript en \mathbb{K} spécifie complètement les nombres flottants, comme il est d'usage en \mathbb{K} . Dans les sémantiques squelettiques, il est au contraire courant d'abstraire les flottants, de façon à pouvoir implémenter des versions

concrètes et abstraites adaptées de leurs opérations. Ce choix de degré d'abstraction dans un tel compilateur vers \mathbb{K} me paraît important et peut amener à des développements intéressants.

Compilation à coûts prédictifs Les systèmes embarqués sont souvent soumis à des contraintes fortes en termes de coûts (en mémoire, énergie, ou temps d'exécution). Il est donc important que les programmes s'exécutant sur ces derniers soient associés à des coûts prédictifs. De tels compilateurs existent déjà [ARG10], mais ils sont associés à un langage source particulier, souvent bas-niveau. Les généraliser dans le cadre des sémantiques squelettiques permettrait de construire un compilateur général prenant en compte ces coûts, y compris pour des langages complexes tels JavaScript. Un tel compilateur pourra être relié avec les analyses en coûts et en énergie proposés dans le premier axe : une telle preuve semble en effet s'inscrire dans le mécanisme de preuve de cohérence d'interprétations squelettiques.

Compilation générale Les langages cibles des différents compilateurs décrits ci-dessus ont en général une sémantique assez simple. Il est ainsi envisageable de construire une sémantique squelettique pour ces langages. Compiler une sémantique squelettique vers cet autre langage, puis composer la sémantique squelettique associée à ce même langage permettrait de produire une nouvelle sémantique squelettique très différente de la sémantique initiale. On a donc une transformation de sémantiques squelettiques qui modifie complètement leur structure. Les sémantiques squelettiques ayant été conçues pour faciliter les preuves, cela semble indiquer qu'il existe un moyen d'exprimer dans le cas général le comportement d'un compilateur, sans sortir du formalisme des sémantiques squelettiques. Un tel résultat permettrait sur le long terme de guider la preuve d'un compilateur, y compris pour des langages à la sémantique complexe. À plus court terme, les transformations de sémantiques squelettiques peuvent s'avérer utiles : j'ai déjà mis en avant des règles de réécriture sur ces sémantiques qui ne modifient pas la sémantique concrète, mais améliorent la précision de la sémantique abstraite. Une étude approfondie de telles règles de réécriture apporterait de nouveaux outils théoriques au formalisme.

4. Application des sémantiques squelettiques

Les sémantiques squelettiques étant assez jeunes, elles sont restées pour l'instant assez théoriques. J'aimerais appliquer ce formalisme dans des cas concrets. Ces cas d'applications permettront d'évaluer mes travaux dans des systèmes réels.

WebAssembly Je travaille actuellement à une formalisation en Coq de WebAssembly [Wat18]. Je souhaite que cette formalisation soit entre autres exprimée sous la forme d'une sémantique squelettique. WebAssembly est en effet assez différent des langages considérés jusque là pour les sémantiques squelettiques, étant basé sur une machine à pile. WebAssembly est de plus un langage à la complexité assez faible, ce qui facilite l'expérimentation. Ceci permettra de tester l'expressivité des sémantiques squelettiques dans un cadre différent, et donc d'évaluer la pertinence de ces dernières.

Interpréteurs monadiques Les projets JSCert et CoqR ont tout deux un interpréteur monadique. Dans le cadre du projet JSExplain [CSW18], il a été montré qu'un tel interpréteur peut être reformulé de plusieurs manières différentes mais équivalentes. J'aimerais donc construire un compilateur de tels interpréteurs monadiques vers des sémantiques squelettiques, à la manière de JSExplain. Ce projet conduira ainsi à moyen terme à une sémantique squelettique correspondant à JSCert et CoqR, ce qui permettra à plus long terme d'appliquer les résultats de l'article original [Bod+19] (notamment sur l'interprétation abstraite) à des langages complexes comme R.

Systèmes temps-réels Toujours afin de tester l'expressivité des sémantiques squelettiques, je compte formaliser un langage synchrone, par exemple Lustre, dans le cadre des sémantiques squelettiques. Un tel projet de formalisation pourra se faire en se basant sur un interpréteur monadique. L'analyse de programmes synchrones étant très différente de l'analyse de programmes plus classiques, cette direction de recherche aboutira naturellement à des analyses spécifiques pour les sémantiques squelettiques de tels langages. En particulier, la compilation d'un langage synchrone vient avec beaucoup de contraintes en terme de temps d'exécution : cette direction de recherche pourra servir de cas d'application pour la compilation de sémantiques squelettiques sous contraintes.

5. Intégration dans Spades

Je pense que les sémantiques squelettiques, de part leur capacité à formaliser des langages très complexes tout en assurant une grande capacité d'analyse, fournissent une approche différente de la conception de langages de programmation. Je pense en particulier qu'il est possible de les utiliser pour concevoir des langages de programmation exprimant *conjointement* la sémantique et le temps d'exécution des programmes, ce qui est un des objectifs de l'équipe Spades.

L'étude des coûts en temps et en énergie [And+13; WRG16] est un des thèmes de recherche de l'équipe qui m'intéresse beaucoup. En particulier, l'expertise d'Alain Girault [ARG10] en compilation sous contraintes sera très précieuse pour construire des compilations squelettiques à coûts prédictifs. Les analyses génériques de temps d'exécution de Pascal Fradet et Sophie Quinton [Fra+18] m'intéressent aussi, et il me semble intéressant de lier ses formalisations Coq avec celle des sémantiques squelettiques.

J'apporte enfin à l'équipe mon expertise Coq, qui vient compléter l'expertise locale.

Bibliographie

- [AGD05] Christopher Anderson, Paola Giannini, and Sophia Drossopoulou. “Towards Type Inference for JavaScript”. In: *ECOOP*. 2005.
- [And+13] Sidharta Andalam, Roopak Sinha, Partha Roop, Alain Girault, and Jan Reineke. “Precise Timing Analysis for Direct-Mapped Caches”. In: *Design Automaton Conference, DAC*. Austin, TX, United States: ACM, June 2013. URL: <https://hal.inria.fr/hal-00842368>.
- [ARG10] Sidharta Andalam, Partha Roop, and Alain Girault. “Predictable Multithreading of Embedded Applications using PRET-C”. In: *MEMOCODE*. 2010.
- [BCO05] Josh Berdine, Cristiano Calcagno, and Peter W. O’Hearn. “Smallfoot: Modular Automatic Assertion Checking with Separation Logic”. In: *International Symposium on Formal Methods for Components and Objects*. 2005.
- [BDT18] Martin Bodin, Tomás Diaz, and Éric Tanter. “A Trustworthy Mechanized Formalization of R”. In: *DLS*. 2018. URL: <https://hal.archives-ouvertes.fr/hal-02409674>.
- [Bes+10] Frédéric Besson, Thomas Jensen, David Pichardie, and Tiphaine Turpin. “Certified result checking for polyhedral analysis of bytecode programs”. In: *International Symposium on Trustworthy Global Computing*. Springer. 2010, pp. 253–267.
- [BJS13] Martin Bodin, Thomas Jensen, and Alan Schmitt. “Pretty-big-step-semantics-based Certified Abstract Interpretation (Preliminary version)”. In: *Festschrift for David Schmidt*. 2013. URL: <https://arxiv.org/abs/1309.5149>.
- [BJS14] Martin Bodin, Thomas Jensen, and Alan Schmitt. “Pretty-big-step-semantics-based Certified Abstract Interpretation”. In: *JFLA*. 2014. URL: <https://hal.inria.fr/hal-01111588>.
- [BJS15] Martin Bodin, Thomas Jensen, and Alan Schmitt. “Certified Abstract Interpretation with Pretty-Big-Step Semantics”. In: *CPP*. 2015. URL: <https://hal.inria.fr/hal-01111588>.
- [BJS16] Martin Bodin, Thomas Jensen, and Alan Schmitt. “An Abstract Separation Logic for Interlinked Extensible Records”. In: *JFLA*. 2016. URL: <https://hal.archives-ouvertes.fr/hal-01333600>.
- [Bod+14] Martin Bodin, Arthur Charguéraud, Daniele Filaretti, Philippa Gardner, Sergio Maffeis, Daiva Naudžiūnienė, Alan Schmitt, and Gareth Smith. “A Trusted Mechanised JavaScript Specification”. In: *POPL (2014)*. URL: <https://hal.inria.fr/hal-00910135>.
- [Bod+19] Martin Bodin, Philippa Gardner, Thomas Jensen, and Alan Schmitt. “Skeletal Semantics and their Interpretations”. In: *POPL (2019)*. URL: <https://doi.org/10.1145/3290357>.
- [Bod16] Martin Bodin. “Sémantique et analyse certifiées de JavaScript”. PhD thesis. 2016. URL: <https://tel.archives-ouvertes.fr/tel-01478722>.
- [Bod18] Martin Bodin. “A Coq Formalisation of a Core of R”. In: *CoqPL*. 2018. URL: <https://popl18.sigplan.org/details/CoqPL-2018/3/A-Coq-Formalisation-of-a-Core-of-R>.
- [BR15] Denis Bogdanas and Grigore Roşu. “K-Java: a Complete Semantics of Java”. In: *POPL*. 2015.
- [BS13] Martin Bodin and Alan Schmitt. “A Certified JavaScript Interpreter”. In: *JFLA*. 2013. URL: <https://hal.inria.fr/hal-00779459>.
- [Bur11] Patrick Burns. *The R Inferno*. 2011.
- [CHP+84] Thierry Coquand, Gérard Huet, Christine Paulin, et al. *the Coq Proof Assistant*. 1984. URL: <https://coq.inria.fr/>.
- [Chu+15] Martin Churchill, Peter D Mosses, Neil Sculthorpe, and Paolo Torrini. “Reusable components of semantic specifications”. In: *Transactions on Aspect-Oriented Software Development XII*. 2015.
- [CS10] Michael R Clarkson and Fred B Schneider. “Hyperproperties”. In: *Journal of Computer Security* 18.6 (2010).
- [CSW18] Arthur Charguéraud, Alan Schmitt, and Thomas Wood. “JSExplain: A Double Debugger for JavaScript”. In: *Companion Proceedings of the The Web Conference 2018*. International World Wide Web Conferences Steering Committee. 2018.
- [Fra+17] José Fragoso Santos, Petar Maksimović, Daiva Naudžiūnienė, Thomas Wood, and Philippa Gardner. “JaVerT: JavaScript verification toolchain”. In: *POPL (2017)*.
- [Fra+18] Pascal Fradet, Maxime Lesourd, Jean-François Monin, and Sophie Quinton. “A Generic Coq Proof of Typical Worst-Case Analysis”. In: *RTSS*. 2018.
- [Gar+19] Philippa Gardner, José Fragoso Santos, Maksimović Petar, and Sacha-Élie Ayoun. *Gillian: A General Static Analysis Framework based on Separation Logic*. ECOOP Summer School. 2019.
- [GMS12] Philippa Gardner, Sergio Maffeis, and Gareth Smith. “Towards a Program Logic for JavaScript”. In: *POPL*. 2012.

- [GNS13] Philippa Gardner, Daiva Naudžiūnienė, and Gareth Smith. “JuS: Squeezing the Sense out of JavaScript Programs”. In: *JSTools@ECOOP*. 2013.
- [GV19] Aviral Goel and Jan Vitek. “On the Design, Implementation, and Use of Laziness in R”. In: *OOPSLA* (2019).
- [HAP14] Thomas Herndon, Michael Ash, and Robert Pollin. “Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff”. In: *Cambridge journal of economics* 38.2 (2014).
- [IO01] Samin Ishtiaq and Peter O’Hearn. “BI as an Assertion Language for Mutable Data Structures”. In: *POPL* (2001).
- [Jou+15] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. “A Formally-Verified C Static Analyzer”. In: *POPL*. 2015. URL: <https://hal.inria.fr/hal-01078386>.
- [Jun+17a] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. “RustBelt: Securing the foundations of the Rust programming language”. In: *POPL* (2017).
- [Jun+17b] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. “Iris from the Ground Up”. In: *Journal of Functional Programming* (2017).
- [Ler+08] Xavier Leroy, Sandrine Blazy, Zaynah Dargaye, and Jean-Baptiste Tristan. *CompCert: Compilers you can formally trust*. 2008. URL: <http://compcert.inria.fr/>.
- [LG18] Liyi Li and Elsa L Gunter. *Isak: A complete semantics of \mathbb{K}* . Tech. rep. 2018.
- [Mar+14] André Maroneze, Sandrine Blazy, David Pichardie, and Isabelle Puaut. “A Formally Verified WCET Estimation Tool”. In: *WCET*. 2014.
- [Mem+16] Kayvan Memarian, Justus Matthiesen, James Lingard, Kyndylan Nienhuis, David Chisnall, Robert NM Watson, and Peter Sewell. “Into the Depths of C: Elaborating the de facto Standards”. In: *PLDI* (2016).
- [MMT09] Sergio Maffei, John C. Mitchell, and Ankur Taly. “Isolating JavaScript with Filters, Rewriting, and Wrappers”. In: *ESORICS*. 2009.
- [MMT10] Sergio Maffei, John C. Mitchell, and Ankur Taly. “Object Capabilities and Isolation of Untrusted Web Applications”. In: *SP*. IEEE, 2010.
- [Mos05] Peter D Mosses. *Action semantics*. Vol. 26. Cambridge University Press, 2005.
- [MT09] Sergio Maffei and Ankur Taly. “Language-Based Isolation of Untrusted JavaScript”. In: *CSF*. IEEE. 2009.
- [NMR03] Hemendra Singh Negi, Tulika Mitra, and Abhik Roychoudhury. “Accurate Estimation of Cache-related Preemption Delay”. In: *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2003, pp. 201–206.
- [Pol+12] Joe Gibbs Politz, Matthew J. Carroll, Benjamin S. Lerner, Justin Pombrio, and Shriram Krishnamurthi. “A Tested Semantics for Getters, Setters, and eval in JavaScript”. In: *DLS* (2012).
- [PSR15] Daejun Park, Andrei Stefanescu, and Grigore Roşu. “KJS: A Complete Formal Semantics of JavaScript”. In: *PLDI*. 2015.
- [Ric+11] Gregor Richards, Christian Hammer, Brian Burg, and Jan Vitek. “The eval that Men Do. A Large-Scale Study of the Use of eval in JavaScript Applications”. In: *ECOOP*. 2011.
- [RŞ10] Grigore Roşu and Traian Florin Şerbănuţă. “An Overview of the \mathbb{K} Semantic Framework”. In: *Journal of Logic and Algebraic Programming* (2010).
- [Sew+10] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strnisa. “OTT: Effective Tool Support for the Working Semanticist”. In: *Journal of Functional Programming* (2010).
- [tea13] The Tessel team. *Tessel 2*. 2013. URL: <https://tessel.io/>.
- [tea15] The KinomaJS team. *A JavaScript runtime optimized for the applications that power IoT devices*. 2015. URL: <https://github.com/Kinoma/kinomajs>.
- [TFW00] Henrik Theiling, Christian Ferdinand, and Reinhard Wilhelm. “Fast and Precise WCET Prediction by Separated Cache and Path Analyses”. In: *Real-Time Systems* (2000).
- [Thi05] Peter Thiemann. “Towards a Type System for Analyzing JavaScript Programs”. In: *ESOP*. 2005.
- [Wat18] Conrad Watt. “Mechanising and verifying the WebAssembly specification”. In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM. 2018, pp. 53–65.
- [Wil12] Gordon Williams. *Espruino - JavaScript for Microcontrollers*. 2012. URL: <https://www.espruino.com/>.
- [WRG16] Jiajie Wang, Partha S Roop, and Alain Girault. “Energy and Timing Aware Synchronous Programming”. In: *International Conference on Embedded Software, EMSOFT’16*. Pittsburgh, United States: ACM, Oct. 2016, p. 10. URL: <https://hal.inria.fr/hal-01412100>.
- [Zap] Carlo Zapponi. *GitHut, A small place to discover languages in GitHub*. URL: <https://githut.info/> (visited on Feb. 14, 2020).

Formulaire 5 — LISTE COMPLÈTE DES CONTRIBUTIONS

Form 5 — COMPLETE LIST OF CONTRIBUTIONS

1. Publications caractéristiques / *Representative publications*

- [BDT18] Martin Bodin, Tomás Diaz, and Éric Tanter. “A Trustworthy Mechanized Formalization of R”. In: *DLS*. 2018. URL: <https://hal.archives-ouvertes.fr/hal-02409674>.
- [Bod+14] Martin Bodin, Arthur Charguéraud, Daniele Filaretti, Philippa Gardner, Sergio Maffei, Daiva Naudžiūnienė, Alan Schmitt, and Gareth Smith. “A Trusted Mechanised JavaScript Specification”. In: *POPL* (2014). URL: <https://hal.inria.fr/hal-00910135>.
- [Bod+19] Martin Bodin, Philippa Gardner, Thomas Jensen, and Alan Schmitt. “Skeletal Semantics and their Interpretations”. In: *POPL* (2019). URL: <https://doi.org/10.1145/3290357>.

2. Publications

Dans mon domaine, l’ordre des auteurs est la plupart du temps l’ordre alphabétique.

2.1 Revues internationales / *International journals*

2.2 Conférence internationales avec comité de lecture / *Reviewed international conferences*

- [MB11] David Monniaux and Martin Bodin. “Modular Abstractions of Reactive Nodes using Disjunctive Invariants”. In: *APLAS*. 2011. URL: [10.1007/978-3-642-25318-8_5](https://doi.org/10.1007/978-3-642-25318-8_5).
- [BJS13] Martin Bodin, Thomas Jensen, and Alan Schmitt. “Pretty-big-step-semantics-based Certified Abstract Interpretation (Preliminary version)”. In: *Festschrift for David Schmidt*. 2013. URL: <https://arxiv.org/abs/1309.5149>.
- [EHB13] Jörg Endrullis, Dimitri Hendriks, and Martin Bodin. “Circular Coinduction in Coq Using Bisimulation-Up-To Techniques”. In: *ITP*. 2013. URL: https://doi.org/10.1007/978-3-642-39634-2_26.
- [Bod+14] **Martin Bodin, Arthur Charguéraud, Daniele Filaretti, Philippa Gardner, Sergio Maffei, Daiva Naudžiūnienė, Alan Schmitt, and Gareth Smith.** “A Trusted Mechanised JavaScript Specification”. In: *POPL* (2014). URL: <https://hal.inria.fr/hal-00910135>.
- [BJS15] Martin Bodin, Thomas Jensen, and Alan Schmitt. “Certified Abstract Interpretation with Pretty-Big-Step Semantics”. In: *CPP*. 2015. URL: <https://hal.inria.fr/hal-01111588>.
- [Bod18] Martin Bodin. “A Coq Formalisation of a Core of R”. In: *CoqPL*. 2018. URL: <https://popl18.sigplan.org/details/CoqPL-2018/3/A-Coq-Formalisation-of-a-Core-of-R>.
- [BDT18] Martin Bodin, Tomás Diaz, and Éric Tanter. “A Trustworthy Mechanized Formalization of R”. In: *DLS*. 2018. URL: <https://hal.archives-ouvertes.fr/hal-02409674>.
- [Bod+19] **Martin Bodin, Philippa Gardner, Thomas Jensen, and Alan Schmitt.** “Skeletal Semantics and their Interpretations”. In: *POPL* (2019). URL: <https://doi.org/10.1145/3290357>.

2.3 Livres et chapitres de livre / *Books and book chapters*

2.4 Autres publications internationales (posters, articles courts) / *Other international publications (posters, short papers)*

- [Bod19a] Martin Bodin. *A Trustworthy Mechanized Formalization of R*. Formal Methods for Statistical Software (FMfSS). 2019. URL: <https://samate.nist.gov/FMSwVRodeo/FMfSS2019.html>.
- [Bod19b] Martin Bodin. *Skeletal Semantics*. Poster during the Verified Software Workshop. 2019. URL: <https://www.doc.ic.ac.uk/~mbodin/esplorado/Verified-Software-Workshop-2019.pdf>.

2.5 Revues nationales / *National journals*

2.6 Conférences nationales avec comité de lecture / *Reviewed national conferences*

- [BS13] Martin Bodin and Alan Schmitt. “A Certified JavaScript Interpreter”. In: *JFLA*. 2013. URL: <https://hal.inria.fr/hal-00779459>.

- [BJS14] Martin Bodin, Thomas Jensen, and Alan Schmitt. “Pretty-big-step-semantics-based Certified Abstract Interpretation”. In: *JFLA*. 2014. URL: <https://hal.inria.fr/hal-01111588>.
- [BJS16] Martin Bodin, Thomas Jensen, and Alan Schmitt. “An Abstract Separation Logic for Interlinked Extensible Records”. In: *JFLA*. 2016. URL: <https://hal.archives-ouvertes.fr/hal-01333600>.

2.7 Rapports de recherche et articles soumis / *Research reports and publications under review*

3. Développements technologiques : logiciel ou autre réalisation / *Technology development : software or other realization*

Toutes les réalisations listées ici sont des formalisations Coq en partie extraite vers OCaml. J’ai attaché une grande importance à ce que l’on puisse tester chacune de ces formalisations, et elles sont donc toutes associées à un parseur en OCaml, ainsi que tout ce qui est nécessaire pour les exécuter en pratique. J’attache une grande importance à la documentation en Coq, ce qui a d’ailleurs été remarqué dans mes reviews de l’artéfact de la formalisation des sémantiques squelettiques.

JSCert

- Formalisation associée à [Bod+14], <http://jscert.org/>.
- Software: A-3, SO-3, SM-2, EM-1, SDL-4.
- Own contribution: DA-2, CD-3, MS-1, TPM-1.

Formalisation Coq des sémantiques pretty-big-step

- Formalisation associée à [BJS15], <http://ajacs.inria.fr/coq/cpp2015/>.
- Software: A-2, SO-4, SM-1, EM-1, SDL-4.
- Own contribution: DA-4, CD-4, MS-1, TPM-1.

CoqR

- Formalisation associée à [BDT18], <https://github.com/Mbodin/CoqR>.
- Software: A-3, SO-3, SM-2, EM-2, SDL-4.
- Own contribution: DA-4, CD-4, MS-2, TPM-4.

Formalisation Coq des sémantiques squelettiques

- Formalisation associée à [Bod+19], <https://gitlab.inria.fr/skeletons/Coq>.
- Software: A-3, SO-4, SM-2, EM-1, SDL-4.
- Own contribution: DA-3, CD-4, MS-1, TPM-3.

4. Impact socio-économique et transfert / *Socio-economic impact and transfer*

Les pages suivantes contiennent les documents suivants :

- Les rapports de thèse, de Roberto Giacobazzi et Anders Møller (page 18),
- Le rapport de soutenance de thèse (page 24),
- Une copie du diplôme de thèse (page 26).

Il était aussi demandé de joindre une photographie du candidat de manière facultative :



Martin Bodin

PhD Thesis: Referee Report

Title: **Certified Semantics and Analysis of JavaScript**

Author: **Martin Bodin**

The Thesis attacks the problem of bridging formal methods (e.g., proof-system assisted verification, certified compilation and analysis) with a highly anarchical, complex, and dynamic language such as JavaScript. The relevance of bridging this gap is huge and the thesis courageously attacks this extremely difficult and hot topic. After presenting the language and its memory model, the author introduced the basics of abstract interpretation and theorem proving via Coq. These are the two main techniques used in the thesis for the design of a certified analysis tool for this language. The tool, called JSCert, is presented in all details. The thesis ends with an instantiation of the analyser with a subset of JavaScript that proves its effectivity. The thesis is well written and appears to me technically sound. The whole construction is formalised and validated in Coq.

Chapter 1. This chapter introduces the language and provides an in-depth analysis of its features. I have found this chapter very well written and easy to access also to non specialists! The analysis is deep and shows a good knowledge of the field.

Chapter 2. This chapter provides a formal specification of JavaScript in Coq. After an analysis of Coq-like specifications for other languages (and also a brief introduction to operational semantics in small/big step for λ -calculus which I have found not necessary in the context of this work but anyway interesting) the author analyses the literature in the field of formal semantics of JavaScript programs. JSCert is then introduced. The presentation is complete and full of intuitive ideas to make an often arid presentation of Coq clauses more readable. The huge (900 rules) specification is of course missing in the main document. Section 2.10 compares different solutions to the definition of the semantics of JavaScript in a very detailed way.

Chapter 3. This chapter introduces abstract interpretation. The way this theory is presented is from the perspective of a Coq expert. I liked this angle even if there are different ways for presenting these concepts. Of course the way abstract interpretation is presented is coherent with the focus of the thesis. A missing point is the lack of widening operators in the presentation of the theory. Indeed the domains presented are quite simple and almost toy abstract domains but effective for presenting the main ideas. Of course the main focus of the thesis is in dominating the complexity of a JavaScript certified analyser, and not in dominating the complexity of specific (possibly new) abstract domains. From this perspective I have found Chapter 3 quite well presented and structured.

Chapter 4. This chapter lifts the theory of abstract interpretation (a simplification of this theory I would say) to the building of a static analyser, systematically from a Coq specification of the semantics of the language. This is nowadays well known. The main contribution of the chapter is in handling and dominating the size of the specification, what the author calls "a Large Semantics". The idea of mechanising pretty big-step semantics is indeed interesting. The result is an interesting bridge between abstract and concrete rules where abstract rule definitions and their proofs can be obtained within the same framework. The correctness of the abstract semantics is formally proved. The framework is parametric in the abstract domain and in the language, provided that the language is specified in a pretty big-style operational semantics.

Chapter 5. This chapter deals with the extension of the framework with non-structural rules. These rules allow the introduction of dynamic elements in the static analysis, such as the fix-point extrapolation (widening). Glue-weaken and Trace Partitioning are considered. These are quite common strategies in abstract interpretation based program analyses. The value here is in the formalisation of these rules within this complex framework. I appreciated the way the chapter has been written.

Chapter 6. This chapter introduces separation logic for JavaScript as an abstract domain. This is the most interesting and innovative part of the thesis I think. The whole construction is technically sound and very well detailed, with lot of examples and pictures that show the mechanics of this (more) complex analysis. The notion of *membrane* was, from my point of view, very interesting!

As a global picture, Martin Bodin's thesis is an excellent work, showing how it is possible to dominate and handle the complexity of a highly complex language like JavaScript within tools based on formal methods. A huge work has been done in considering the difficulty of JavaScript both from the point of view of semantics and analysis.

More specific criticisms are described below relatively to *correctness*, *originality*, *significance*, and *quality of presentation*.

Correctness and technical development

The whole approach to the definition of JSCert is sound. The thesis is fully formalised in all details. The choice of specifying and deriving analysers in the framework of abstract interpretation allows the author to formally prove relevant results concerning: correctness and relative abstraction with respect to other well known abstract domains. The only weak aspect is the instantiation. Until Chapter 6 this is done for toy abstract domains. The

abstraction in Chapter 6 instead is far more structured, but applied to a restricted language O'WHILE. The resulting abstraction is anyway definitively non-trivial. However, the focus of the thesis is in managing a complex language like JavaScript with abstract interpretation within a theorem prover, and not in the design of new abstract domains. This perfectly justifies a positive evaluation of the technical achievements of this thesis.

Originality

The idea of the thesis is in the path with previous works by the same group. This is justified by a good record of results and successes in certified program analysis. The case of JavaScript is very interesting from this perspective because it is an horribly complex programming language. The huge effort made in the thesis to formally specify its semantics is remarkable.

Significance

The problem addressed by the thesis is definitively relevant, and well known in the field of static program analysis. The results obtained represent a significant step forward in order to understand how deriving sound-by-construction static analysers of complex (real) programming languages. The author succeeds in addressing these important issues.

Quality of presentation

The presentation is good, and clear in all details. The author have a good perspective of the field, he proved to be mature in the way of making research and he includes all the relevant literature in the bibliography. The first part is very nice and clear in the way the problem is addressed and how the solutions are proposed. All the proposed solutions to the problem of designing a certified program analysis tool for JavaScript are detailed with examples and related works. All the relevant literature is cited in the references with a deep discussion on how it relates to the solutions proposed by the author. The notation is fairly readable even for non specialists, with a minimal knowledge on abstract interpretation, JavaScript and Coq. It is definitively a very well presented PhD thesis for any specialist in the field. In the final version the author should make a more careful spell check as few flaws are present in the text.

Sincerely



Prof. Roberto Giacobazzi
U. of Verona
Italy



DEPARTMENT OF
COMPUTER SCIENCE

AARHUS UNIVERSITY

Report on the PhD dissertation of Martin Bodin

The work presented in Bodin's dissertation "Certified Semantics and Analysis of JavaScript" shows that pretty-big-step semantics and Coq can be used for formalizing both concrete and abstract semantics for JavaScript. Formalizing such a highly complex programming language is a major endeavor. The dissertation is based on material that has been published at the Symposium on Principles of Programming Languages (POPL 2014) and at the Conference on Certified Programs and Proofs (CPP 2015), as well as some unpublished material.

Department of
Computer Science

Anders Møller

Associate Professor, PhD

Date: 5 October 2016

Direct Tel.: +45 2330 9994
Email: amoller@cs.au.dk

Web:
<http://cs.au.dk/~amoller/>

Sender's CVR no.: 31119103

Chapter 1 provides background on the JavaScript language, specifically ECMAScript 5. This chapter gives an introductory overview of the language. Some aspects of the language, in particular the peculiarities of getters, setters, and property attributes, have been ignored, whereas details about parsing and entertaining but perhaps less relevant facts about implicit type conversion receive a lot of attention.

Chapter 2 presents the JSCert project, which aims to provide a Coq-based formalization of the JavaScript language semantics. An important design decision has been to use pretty-big-step-style semantics. Bodin argues that this style of semantics has nice properties: local knowledge is enough to know whether a rule applies, and rule duplication is avoided. Besides JSCert, a JavaScript interpreter named JSRef has been developed and proven correct with respect to JSCert. The overall goal is to provide a trustworthy formalization. To this end, JSCert has been developed by following the (precise but informal) ECMAScript language specification, and JSRef has been thoroughly tested using an existing, extensive test suite. This methodology is interesting, yet some questions remain. First, the pretty-big-step semantic rules are far from being a direct transcription of the corresponding parts of the ECMAScript language specification (case in point: Program 2.5 compared to Figure 2.4). Second, it is uncertain whether JSRef has been developed independently of JSCert, which affects the trustworthiness. Bodin uses the term "correctness" meaning that every behavior that is possible according to JSRef is also allowed by JSCert, but the converse property (which Bodin calls "completeness") may not hold. This is a questionable choice of terminology, but it can of course easily be clarified in the final version of the dissertation. As a minor remark, it is surprising that the for-in construct has not been formalized. Even though its semantics is partly implementation-dependent, it ought to be possible to formalize the allowed implementation choices. The last part of the chapter discusses related work, in particular KJS (based on the \mathbb{K}

Department of
Computer Science
Aarhus University
IT-parken, Aabogade 34
DK-8200 Aarhus N
Denmark

Tel.: +45 89425600
Fax: +45 89425601
Email: cs@cs.au.dk
<http://cs.au.dk>



DEPARTMENT OF
COMPUTER SCIENCE

AARHUS UNIVERSITY

framework) and λ_{JS} . A Coq formalization of the latter exists, and the advantages of the JSCert/JSRef approach are entirely clear. Overall, the most interesting result in this part of the dissertation is that pretty-big-step semantics is applicable to formalization of a language as complex as JavaScript.

Chapter 3 provides background on abstract interpretation. Besides building on the classic work by Cousot & Cousot, Bodin uses Schmidt's work on abstract interpretation of big-step semantics and Pichardie's work on formalizing analyses in Coq.

Chapter 4 contains the second main contribution: a framework for expressing pretty-big-step semantics in Coq, together with an approach to build accompanying abstract semantics. This appears to be a powerful technique that others may benefit from when formalizing languages and analyses also for other languages. Starting from the concrete semantics, building abstract semantics is mostly a matter of providing abstract rules for each of the concrete rules, thereby exploiting the locality properties of pretty-big-step semantics. It would be interesting to discuss in more detail to what extent it could be beneficial to automate the construction of the abstract rules, as studied in previous work by Van Horn and Might. This chapter does not involve JavaScript but a simple "while"-language. The abstraction chosen is a basic sign analysis. A proof of correctness of the abstract semantics (using the meaning of "correctness" discussed above; perhaps "soundness" would be a more appropriate term) follows using Schmidt's approach. Notably, the formalization and correctness proof only apply to the abstraction, not to a full abstract interpreter. (That could perhaps have been stated more explicitly earlier in the text.) A section of the chapter discusses certified program verifiers; references to previous work by Pichardie and others would be a welcome addition. The last part of the chapter briefly discusses the work required to connect the approach to JSCert. Not only would that be a significant effort (despite the goal being to minimize the proof effort required to build abstract semantics and certified analyses); it would also be complicated by the fact that this chapter uses a more constrained variant of pretty-big-step semantics than the one used in Chapter 2.

Chapter 5 extends the preceding chapter by considering non-structural rules (i.e., abstract rules that do not follow the structure of the concrete rules): one for supporting widening, and one for a form of trace partitioning. Such rules do not satisfy the restrictions of pretty-big-step, and the correctness theorem from Chapter 4 no longer applies. With the complications that follow, it might have been interesting to consider alternative approaches, for example the AAM approach by Van Horn and Might.

Chapter 6 studies abstract domains for JavaScript analysis. To simplify the development, a small subset is used instead of the full language. The abstraction



DEPARTMENT OF
COMPUTER SCIENCE

AARHUS UNIVERSITY

chosen is based on separation logic and shape analysis. Many JavaScript analysis tools exist already, and the motivation for this particular choice of abstraction is not explained. A notion of membranes is introduced to manage the naming of abstract locations. This is an interesting concept, but unfortunately there is no experimental evaluation of the approach. The chapter concludes with discussions on how the approach could be connected to existing JavaScript program analyzers, in particular TAJIS. Given the very different structure of that tool, it is doubtful whether that would be fruitful.

For the final version of the dissertation, the following minor issues can easily be fixed. It would have been appropriate to clarify the relations to the publications that the dissertation builds on: there is no precise explanation of what parts are reused from the publications and what parts are original contributions. Given the large number of co-authors on the JSCert work, it would also be interesting to see which parts Bodin has focused on. Furthermore, it would have been useful to state explicitly that some chapters and sections (especially Chapters 1 and 3) contain no new results but merely serve as providing background material. Finally, some parts of the text are plagued by typographical and grammatical errors.

In conclusion, Bodin has demonstrated that pretty-big-step semantics and Coq can be used for large-scale programming language formalization, and that such formalizations can serve as useful starting points for building provably correct analyses. Achieving such trustworthiness in a formalization of a language as complex as JavaScript is impressive, and the work raises many interesting questions for future research. For all these reasons I am fully in favor of authorizing Martin Bodin to defend his thesis.

Anders Møller

Université Rennes 1

ATTESTATION DE REUSSITE AU DIPLOME

Le Responsable de la Scolarité atteste que

le doctorat en informatique

a été décerné à


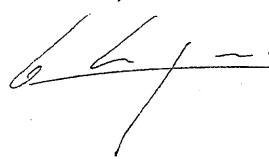
Monsieur MARTIN BODIN

né le 30 décembre 1989 à RENNES (035)

au titre de l'année universitaire 2015/2016 avec la mention Très honorable

Titre des travaux : Certified semantics and analysis of JavaScript
Date de soutenance : 25 novembre 2016
Etablissement soutenance : UNIVERSITE RENNES 1
Jury : Mme SOPHIE PINCHINAT, Président du jury, PROFESSEUR DES UNIVERSITES
UNIVERSITE RENNES 1
Mme PHILIPPA GARDNER, Membre du jury, PROFESSEUR DES UNIVERSITES
M. DANIEL HIRSCHKOFF, Membre du jury, MAITRE DE CONFERENCE
ENS LYON
M. ROBERTO GIACOBAZZI, Rapporteur avant soutenance, PROFESSEUR DES UNIVERSITES
UNIVERSITE DE VERONE
M. THOMAS JENSEN, Directeur de travaux, DIRECTEUR DE RECHERCHES
INRIA Rennes bretagne atlantique
M. ANDERS MOLLER, Rapporteur avant soutenance, PROFESSEUR DES UNIVERSITES
M. ALAN SCHMITT, Co directeur, DIRECTEUR DE RECHERCHES
INRIA Rennes bretagne atlantique
Ecole doctorale : Mathématiques, informatique, signal et électronique et télécommunications.
Formation doctorale : INFORMATIQUE
Section CNU : 27 - Informatique

Fait à Rennes, le 29 novembre 2016



Fabrice LE GOUGUEC

N° étudiant : 13026572

Doctorat de l'Université de Rennes 1 – mention **Informatique**

Soutenu le **25 Novembre 2016** par **BODIN Martin**

RAPPORT DE SOUTENANCE (Signature obligatoire de tous les membres du jury)

Nom du Président de jury: Sophie PINETTINAT


Martin Bodin a présenté ses travaux sur la sémantique et l'analyse certifiées des programmes JavaScript. Il a décrit ses contributions majeures pour la sémantique formelle de JavaScript, et son développement pour la dérivation systématique d'interprétations abstraites basées sur une "pretty-big-step-semantics". Enfin, il a exploré l'applicabilité de sa méthode au domaine abstrait de la logique de séparation.

Le jury a apprécié la clarté de l'exposé au regard de la complexité de la sémantique de JavaScript et des résultats obtenus par le candidat: M. Bodin a su choisir des exemples simples mais significatifs pour transmettre ses idées et défendre sa méthodologie.


En sus, Martin Bodin a parfaitement répondu aux nombreuses questions y compris celles levées par les rapporteurs dans leurs rapports. Ses réponses étaient claires et précises, démontrant ainsi sa maturité et son expertise dans le domaine. Ses réponses ont aussi permis d'identifier des directions de recherche les plus prometteuses émanant de ce travail de thèse. Les membres du jury ont été impressionnés par les résultats de la thèse et par sa soutenance, ce qui leur permet de prédire une carrière scientifique brillante au candidat.

Pour toutes ces raisons, le jury est unanime pour décerner à Monsieur Martin Bodin le titre de Docteur en Informatique, label européen, de l'Université de Rennes 1, avec la mention "Très Honorable".

Signatures des membres du jury :


S. PINETTINAT


Thomas JENSEN

Anders Møller

PHILIPPA GARDNER


Daniel HIRSCHKOPF
Alex SCHNITTI



Ministère de l'éducation nationale, de l'enseignement supérieur et de la recherche

UNIVERSITÉ RENNES 1
DOCTORAT

Vu le code de l'éducation, notamment ses articles L. 612-7, L. 613-1, D. 613-3, et D. 613-6 ;

Vu le code de la recherche, notamment son article L.412-1 ;

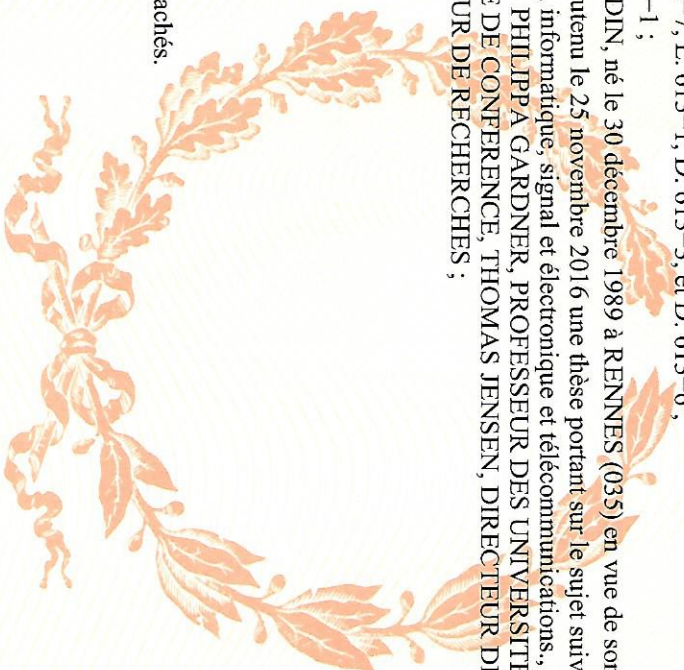
Vu les pièces justificatives produites par M. MARTIN BODIN, né le 30 décembre 1989 à RENNES (035) en vue de son inscription en doctorat ;

Vu le procès-verbal du jury attestant que l'intéressé a soutenu le 25 novembre 2016 une thèse portant sur le sujet suivant : **Certified semantics and analysis of JavaScript** préparée au sein de l'école doctorale Mathématiques, informatique, signal et électronique, devant un jury présidé par SOPHIE PINCHINAT, PROFESSEUR DES UNIVERSITÉS et composé de PHILIPPA GARDNER, PROFESSEUR DES UNIVERSITÉS, ROBERTO GIACOBAZZI, PROFESSEUR DES UNIVERSITÉS, DANIEL HIRSCHKOF, MAITRE DE CONFERENCE, THOMAS JENSEN, DIRECTEUR DE RECHERCHES, ANDERS MOLLER, PROFESSEUR DES UNIVERSITÉS, ALAN SCHMITT, DIRECTEUR DE RECHERCHES ;

Vu la délibération du jury ;

Le diplôme de **DOCTORAT** en informatique
mention très honorable
est décerné à **M. MARTIN BODIN**

au titre de l'année universitaire 2015-2016
et confère le **grade de docteur**,
pour en jouir avec les droits et prérogatives qui y sont attachés.



Fait le 19 décembre 2016

Le titulaire

Le Président

A handwritten signature in black ink, appearing to read 'D. Alis'.

*Le Recteur d'Académie,
Chancelier des universités*

A handwritten signature in black ink, appearing to read 'Thierry TERRET'.

N° **RENNI 12555693**
/2016201508335

David ALIS

Thierry TERRET