Ce document est un résumé en français de l'article “Skeletal Semantics and Their Interpretations” [Bod+19], soit « Les sémantiques squelettiques et leurs interprétations ». Cet article a été publié à POPL (“Principes of Programming Languages”, soit le symposium international sur les principes des languages de programmation), une conférence internationale $A*$, référence dans ce domaine. L'article est accompagné d'un artefact disponible à l'adresse `https://gitlab.inria.fr/skeletons/Coq/tree/POPL2019`. Cet artefact a été évalué par un comité spécifique comme étant réutilisable, ce qui est la meilleur note possible pour un artefact dans cette conférence.

**Contexte** De nombreux langages de programmation utilisés dans l'industrie ont été formalisés dans des assistants de preuve : ML [Owe08], C [Nor98 ; BL09], ainsi que JavaScript [Bod+14]. Ces formalisation ont ensuite été utilisées pour construire et vérifier des analyses de programme [KN03 ; Cac+05 ; Jou16]. Cependant, de telles formalisations demandent de grands efforts pour être effectivement utilisées, et frôlent souvent les limites des assistants de preuve utilisés.

L'interprétation abstraite [CC77] est une théorie très utilisée en analyse de programmes. Elle décrit de manière générale la cohérence d'une sémantique abstraite avec une sémantique concrète, et propose même une méthodologie pour construire de tels sémantiques abstraites dans certains cas pratiques [Cou99 ; MJ09 ; VM11]. En particulier, Schmidt [Sch95 ; Sch97] montre comment construire une sémantique abstraite lorsque la sémantique concrète est fournie sous forme de règles de réduction. Ces travaux sont génériques, mais pas systématiques. Bodin *et al.* [BJS15] ont proposé une approche systématique, mais sur un format de règle spécifique et non-standard — le *pretty-big-step* [Cha13], qui a joué un rôle central dans la formalisation de JavaScript [Bod+14].

De nombreux cadres formels proposent maintenant des langages de spécification [HHP87 ; PS99 ; RŞ10 ; Jun+17], mais très peu ont pour but d'être liés à des analyses de programme. Une exception notable est le cadre d'Iris [Jun+17], qui fournit des outils pour raisonner sur les programmes concurrents. Ce cadre ne fournit cependant pas de théorème de cohérence générique avec une sémantique abstraite.

**Contributions** Cet article propose un nouveau langage de description de sémantiques appelé *sémantique squelettique*. La nouveauté est que ce méta-langage n'est ni concret, ni abstrait. À la place, il est possible de définir de nombreuses interprétations de sémantiques squelettiques, et en particulier une interprétation concrète et une interprétation abstraite. De plus, ce travail fournit un moyen de montrer de manière indépendante de la sémantique squelettiques que deux interprétations sont cohérentes. En conséquence, la preuve que l'interprétation abstraite ainsi construite est cohérente avec l'interprétation concrète est applicable pour toute sémantique squelettique.

Cet article définit quatre interprétations réutilisables pour toute sémantique squelettique :

- Une interprétation de bonne formation. Cette interprétation permet de vérifier qu'une sémantique squelettique donnée respecte des règles de bonne formation et de typage. La plupart des résultats de l'article supposent par la suite que les sémantiques squelettiques considérées sont bien formées.
- L'interprétation concrète, qui correspond à la lecture habituelle des règles de réduction concrètes.
- L'interprétation abstraite, qui généralise l'approche de Schmidt pour une sémantique squelettique. Contrairement aux travaux antérieurs, cette interprétation abstraite est ici systématique : elle s'applique directement à toute sémantique squelettique.
- La génération de contraintes. Cette interprétation construit un des modèles intermédiaires souvent utilisés en interprétation abstraite pour définir la sémantique d'un programme. Elle va ainsi générer un ensemble de contraintes de la forme $[\mathrm{pp}_1\text{-}x_1 \sqsubseteq \mathrm{pp}_2\text{-}x_2]$ pour décrire le fait que la variable $x_2$ au point de programme $\mathrm{pp}_2$ peut hériter des valeurs de la variable $x_1$ au point de programme $\mathrm{pp}_1$. Une analyse de programme peut ensuite être décrite comme une solution à ce système de contrainte. Cette interprétation est moins générale que l'interprétation abstraite, mais elle correspond à ce sur quoi se basent beaucoup de travaux de la littérature du domaine : elle illustre ainsi la généralité des sémantiques squelettiques.

En plus des définitions d'interprétations, cet article prouve les théorèmes de cohérence associés. Tous ces théorèmes sont prouvés à partir d'un théorème central, qui permet de ramener la preuve de cohérence de deux interprétations à la preuve de cohérence de leurs constituants. En particulier on trouvera :

- Les théorèmes de cohérence entre l'interprétation de bonne formation et l'ensemble des autres interprétations de l'article.
- Le théorème de cohérence entre l'interprétation concrète et l'interprétation abstraite.
- Le théorème de cohérence entre l'interprétation abstraite et la génération de contrainte.

Ces théorèmes ont tous été montrés de manière générique, sans supposer une sémantique squelettique particulière. L'article instancie ces théorèmes sur un langage d'exemple, mais les théorèmes s'appliquent donc tout aussi bien sur n'importe quelle autre sémantique squelettique. Ceci est illustré en fin d'article par une extension du langage initialement considéré : il est ainsi montré qu'aucun autre lemme n'a besoin d'être prouvé pour retrouver tous les théorèmes précédemment prouvés sur le langage initial.

L'article montre aussi comment montrer correctes certaines règles dérivées qui sont parfois utilisés en interprétation abstraite. En particulier la règle suivante — habituellement associée aux boucles — et la division de l'état abstrait,

qui permet d'améliorer la précision de l'interprétation abstraite des tests.

$$\frac{\sigma^\# \vdash e : v^\# \qquad \sigma^\# \vdash s : \sigma^\#}{\sigma^\# \vdash \mathit{while}\ e\ s : \sigma^\#}$$

Une formalisation Coq des principaux théorèmes, notamment la cohérence entre l'interprétation abstraite et concrète, ainsi que des exemples est fournie en tant qu'artefact à l'adresse `https://gitlab.inria.fr/skeletons/Coq/tree/POPL2019`. Les sémantiques squelettiques sont ainsi un format de spécification de langages permettant d'obtenir de manière systématique toute une gamme d'interprétations, notamment les interprétations concrètes et abstraites. L'article fournit des moyens de construire et manipuler de telles interprétations en en montrant quatre exemples détaillés.

# Références


[BJS15] Martin Bodin, Thomas Jensen et Alan Schmitt. “Certified Abstract Interpretation with Pretty-Big-Step Semantics”. In : *CPP*. 2015.

[BL09] Sandrine Blazy et Xavier Leroy. “Mechanized Semantics for the Clight Subset of the C Language”. In : *Journal of Automatic Reasoning* (2009).

[Bod+14] Martin Bodin et al. “A Trusted Mechanised JavaScript Specification”. In : *POPL*. 2014.

[Bod+19] Martin Bodin et al. “Skeletal Semantics and their Interpretations”. In : *POPL* (2019).

[Cac+05] David Cachera et al. “Extracting a Data Flow Analyser in Constructive Logic”. In : *Theorical Computer Science* (2005).

[CC77] Patrick Cousot et Radhia Cousot. “Abstract Interpretation : A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In : *POPL*. 1977.

[Cha13] Arthur Charguéraud. “Pretty-Big-Step Semantics”. In : *Programming Languages and Systems*. 2013.

[Cou99] Patrick Cousot. “The Calculational Design of a Generic Abstract Interpreter”. In : *Calculational System Design*. IOS Press, 1999.

[HHP87] Robert Harper, Furio Honsell et Gordon D. Plotkin. “A Framework for Defining Logics”. In : *Proc. of the Symposium on Logic in Computer Science (LICS '87)*. 1987, p. 194-204.

[Jou16] Jacques-Henri Jourdan. “Verasco: a Formally Verified C Static Analyzer”. Thèse de doct. Université Paris Diderot-Paris VII, 2016.

[Jun+17] Ralf Jung et al. “Iris from the Ground Up”. In : *Submitted to JFP* (2017).

[KN03] Gerwin Klein et Tobias Nipkow. “Verified Bytecode Verifiers”. In : *Theorical Computation Sciences* (2003).

[MJ09] Jan Midtgaard et Thomas Jensen. “Control-Flow Analysis of Function Calls and Returns by Abstract Interpretation”. In : *ICFP*. 2009.

[Nor98] Michael Norrish. “Formalising C in HOL”. Thèse de doct. Computer Laboratory, University of Cambridge, 1998.

[Owe08] Scott Owens. “A Sound Semantics for OCamllight”. In : *ESOP*. 2008.

[PS99] Frank Pfenning et Carsten Schürmann. “System Description : Twelf - A Meta-Logical Framework for Deductive Systems”. In : *CADE*. 1999.

[RŞ10] Grigore Roşu et Traian Florin Şerbănuţă. “An Overview of the $\mathbb{K}$ Semantic Framework”. In : *Journal of Logic and Algebraic Programming* (2010).

[Sch95] David A. Schmidt. “Natural-Semantics-Based Abstract Interpretation (preliminary version)”. In : *SAS*. 1995.

[Sch97] David A. Schmidt. “Abstract Interpretation of Small-Step Semantics”. In : *Proceedings of the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*. 1997.

[VM11] David Van Horn et Matthew Might. “Abstracting Abstract Machines : A Systematic Approach to Higher-Order Program Analysis”. In : *Communications of the ACM* (2011).