Méthodes formelles au service de la voiture autonome

Présenté par Vassil Todorov

PSA GROUPE

Direction de la Qualité et de l'Ingénierie

Forum Méthodes Formelles
10 octobre 2017, Toulouse

SOMMAIRE

© PSA Group. All rights reserved.

1

INTRODUCTION

LE DÉPLOIEMENT DU PROGRAMME AVA

"AUTONOMOUS VEHICLE FOR ALL"

INNOVATION BY PSA GROUPE

© PSA Group. All rights reserved.

Le développement logiciel aujourd'hui

- Le logiciel embarqué chez PSA représente 4 domaines :
  - « ADAS »
  - « Contrôle moteur »
  - « Habitacle »
  - « Infotainement »
- Cycle de développement V court : évolutions fréquentes du logiciel
- Architecture AUTOSAR
- Code manuel et généré en langage C/C++
- La norme ISO 26262 n'exige pas de certification du logiciel
- Le conducteur est responsable de la conduite et du contrôle de son véhicule

© PSA Group. All rights reserved.

Conduite autonome : vers une responsabilité du constructeur automobile

- Fonctions de plus en plus critiques et complexes
- Besoin de garanties fortes : le test seul ne suffira plus, usage de méthodes formelles
- La norme ISO 26262 recommande l'usage des méthodes formelles en ASIL C et D :

Table 9 — Methods for the verification of software unit design and implementation

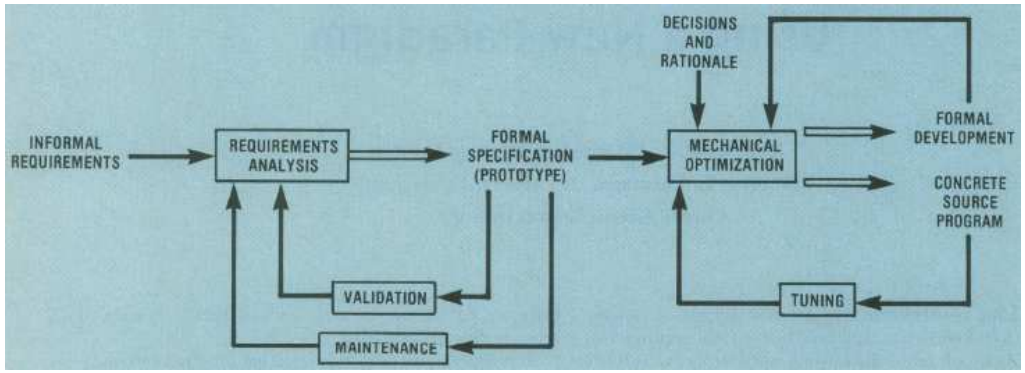
	Methods	ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[b,c]	+	+	++	++
1f	Data flow analysis[b,c]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

- Perspectives de certification

© PSA Group. All rights reserved.

Méthodes formelles et cycle de développement

- Besoin de nouveau cycle de développement
- 1983 : Balzer* propose pour la première fois un tel cycle plaçant les méthodes formelles au cœur du développement (lien entre exigences, prototype, implémentation)



- 2017 : il n'existe toujours pas de cycle de développement standard intégrant des méthodes formelles : chaque société crée le sien
- Nécessité d'appliquer des méthodes formelles sur des cas d'usage de l'automobile pour
  - identifier le périmètre potentiel de leur mise en œuvre
  - identifier les impacts et difficultés à anticiper
  - vérifier quelle méthode pourrait s'intégrer le mieux dans quelle partie du processus

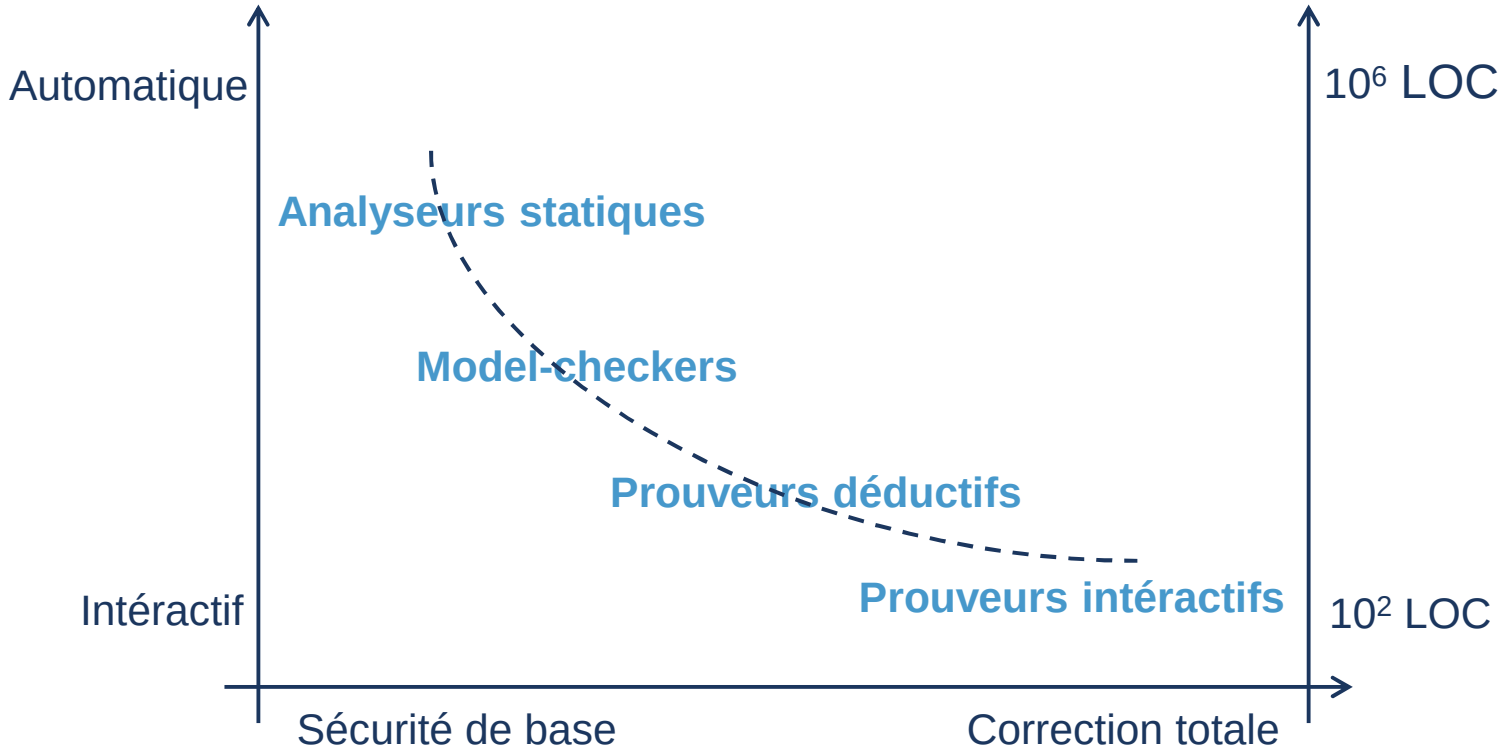
Balzer, R., Jr. T. E. Cheatham, and C. Green. "Software Technology in the 1990's: Using a New Paradigm." Computer 16, no. 11 (November 1983): 39–45. doi:10.1109/MC.1983.1654237.

© PSA Group. All rights reserved.

2

MISE EN ŒUVRE DES MÉTHODES FORMELLES DANS UN CONTEXTE AUTOMOBILE

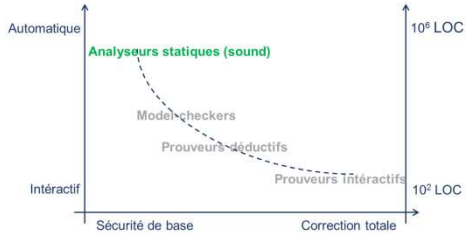
Panorama des outils de vérification formelle



Panorama proposé par Xavier Leroy, INRIA

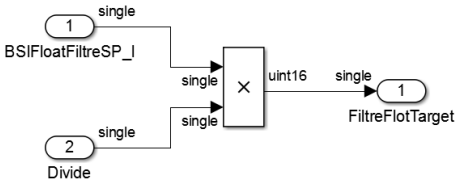
© PSA Group. All rights reserved.

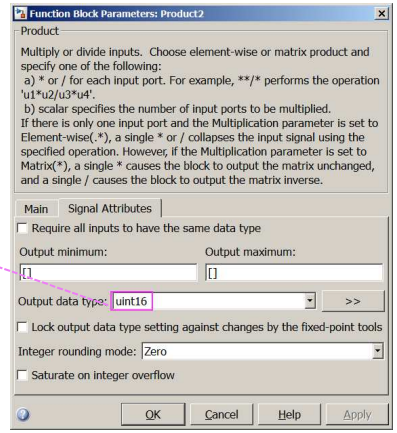
Interprétation abstraite : mise en œuvre



- Expérience avec deux outils sound du marché
- Code C analysé de 273 000 lignes (généré et manuel)
- Progrès des outils : le nombre d'alertes faible
- Détection d'erreurs non vues ni en simulation ni en test :

```
tFLOAT FiltreFlotTarget, BSIFloatFiltreSP_I, Divide;
if (LogicalOperator3_n) {
        FiltreFlotTarget = (tFLOAT)(tWORD)(BSIFloatFiltreSP_I * Divide);
} else {
        FiltreFlotTarget = BSIFloatFiltreSP_I;
}
```





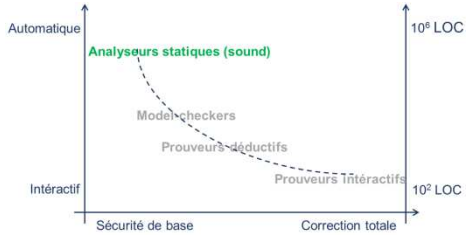
- Surprise : les deux outils ne remontent pas les mêmes alertes

Par exemple :

```
if (a > b) {
        c = 10 / (a - b);
}
```

- comporte une division par 0 pour l'un mais pas pour l'autre

Interprétation abstraite : suite et conclusion



- Qu'est-ce qu'un vrai bug ?

```
typedef unsigned char u8;
void main (void)
{
  u8 a = 1;
  u8 b = ~a;
}
```

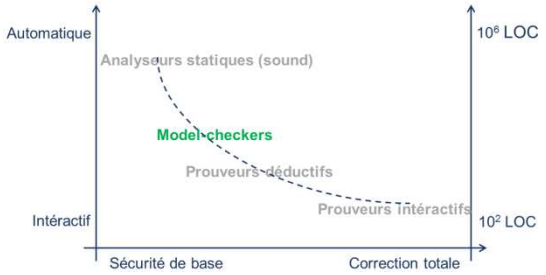
Y a-t-il un overflow dans cet exemple ?

- ✓ Oui, en théorie.
- ✓ Non, en pratique.

- Efficace pour trouver automatiquement des runtime errors et du code mort (problèmes de conception)
- L'indication des plages de valeurs pour chaque flux facilite la compréhension du programme
- Méthode mature et évolutive (grâce aux domaines abstraits extensibles) et peut être introduite dans le processus de développement sans difficultés
- Petit manque dans les outils existants : mode Débutant / mode Expert ainsi que la notion de criticité du code analysé
- Le logiciel critique nécessite d'utiliser plusieurs outils d'analyse statique

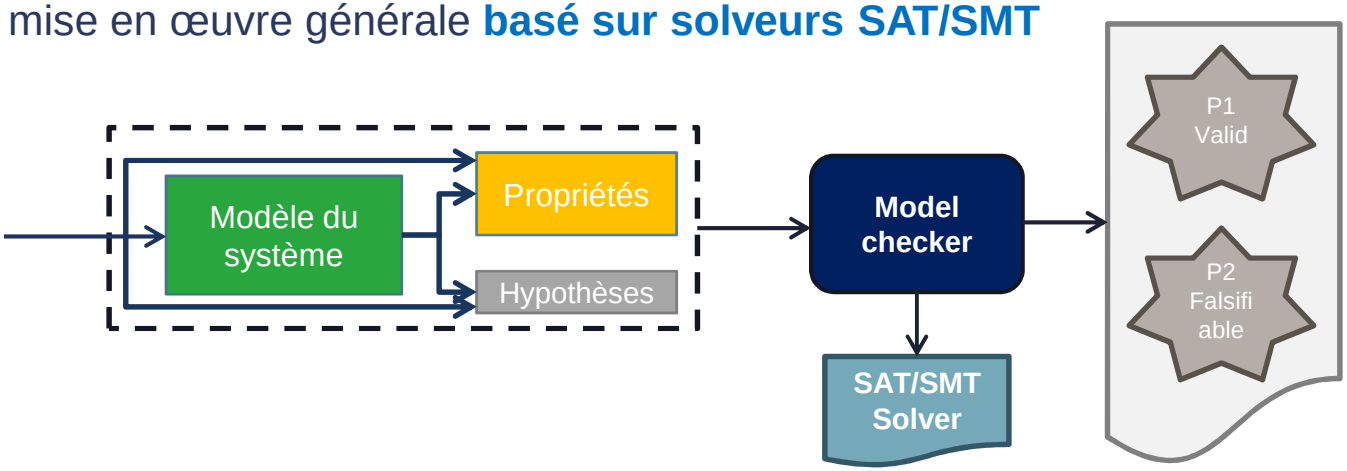
© PSA Group. All rights reserved.

Model checking : mise en œuvre



- Cas d'usage
  - Preuve de propriétés de safety
  - Preuve d'équivalence de modèles
  - Debugging
  - Génération de tests

- Schéma de mise en œuvre générale **basé sur solveurs SAT/SMT**



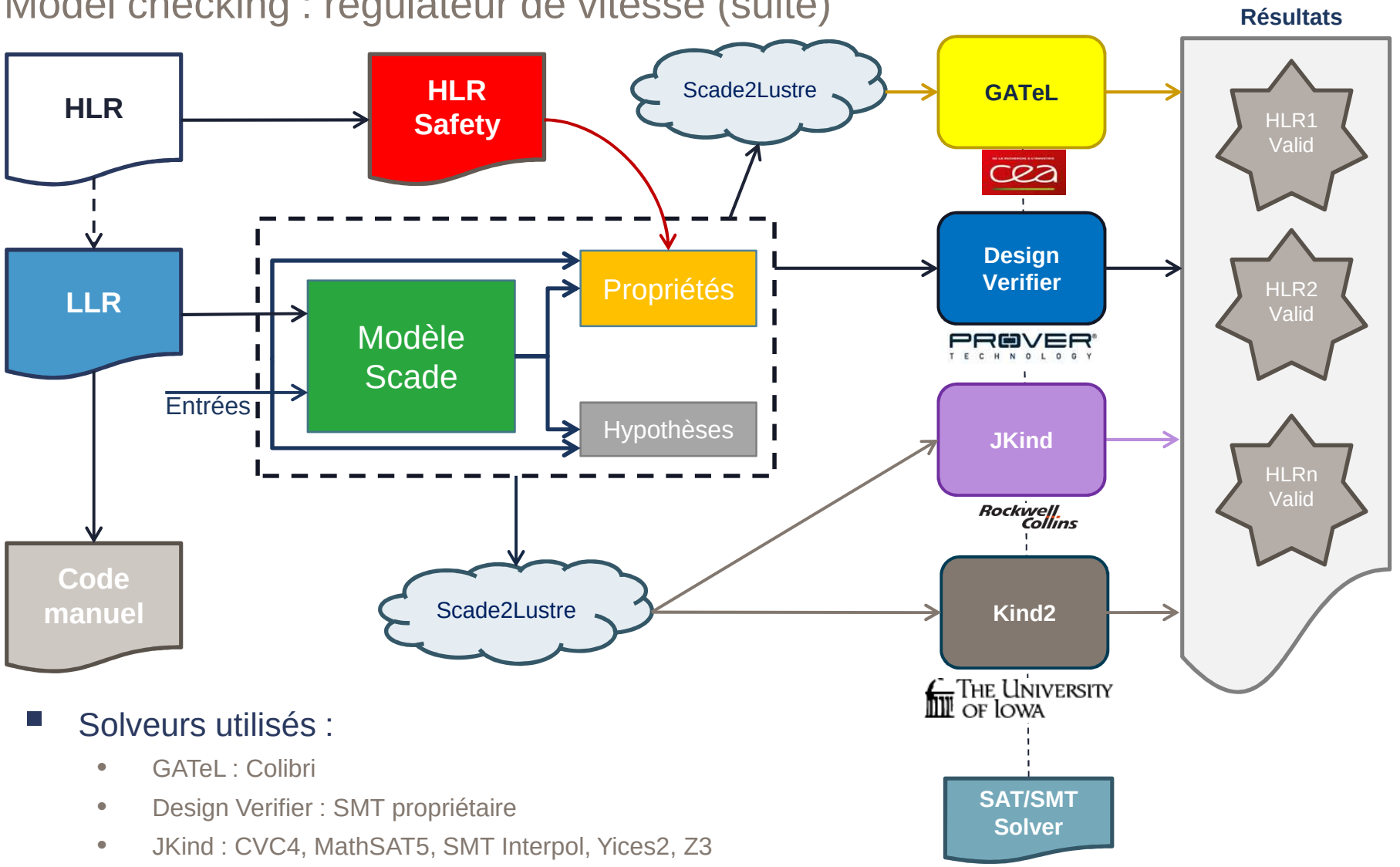
- Possibilité d'avoir un processus certifiable en utilisant un prover certifié qui examine la trace du solveur utilisé

Model checking : Régulateur de vitesse (RVV)

- Choix de SCADE pour sa proximité avec Lustre mais Simulink peut aussi être utilisé :
  - Utilisation des LLR (Low Level Requirement) pour la modélisation
  - 78 nœuds Scade
  - 22 233 lignes de code C (KCG)
  - 2228 lignes de code Lustre
- Modélisation des propriétés de safety HLR (High Level Requirement) en se servant des événements redoutés gravité 4 de sûreté de fonctionnement pour les identifier. Exemple d'exigences vérifiées :

REQ-01 (1.0)	Un appui simple sur le push Cancel/Resume désactive le RVV.
REQ-02 (1.0)	Si la fonction RVV exploite le moteur thermique, la coupure ou calage du moteur désactive le RVV.
REQ-03 (1.0)	La coupure du contact désactive le RVV.
REQ-04 (1.0)	Dans le cas où l'exigence de désactivation de la fonction RVV à partir d'un appui sur la pédale de frein ne respecte pas les objectifs de SDF, une sécurisation fonctionnelle devra être réalisée (pe : désactivation sur décélération véhicule sans appui sur la pédale de frein).
REQ-05 (1.0)	Un défaut de la fonction ou d'un des organes contributeurs (en particulier ceux participants à la tenue d'ERG4) désactive le RVV en moins de T_DESACT_RVV. (Temps entre l'identification du défaut et le passage à l'état INACTIF de la fonction)
REQ-06 (1.0)	La désélection du RVV désactive le RVV en moins de T_DESACT_RVV. (Temps entre l'action conducteur de désélection et le passage à l'état INACTIF de la fonction)
REQ-07 (1.0)	En cas de déclenchement d'un Freinage Automatique, la fonction RVV doit se désactiver en moins de T_DESACT_RVV après déclenchement du freinage automatique.
REQ-08 (1.0)	Une interdiction de régulation faite par les fonctions de sécurités passives désactive le RVV.
REQ-09 (1.0)	Si la vitesse véhicule devient inférieure à VITESSE_RVV_MIN, alors le RVV est désactivé en moins de T_DESACT_RVV.

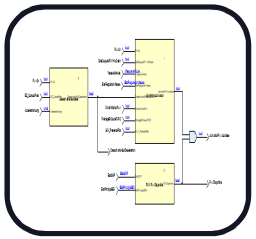
Model checking : régulateur de vitesse (suite)



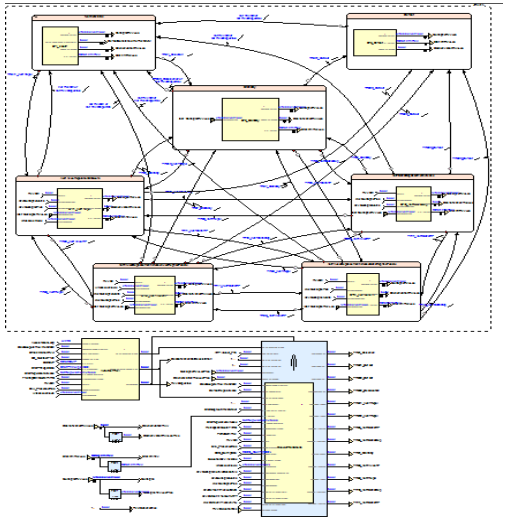
- Solveurs utilisés :
  - GATeL : Colibri
  - Design Verifier : SMT propriétaire
  - JKind : CVC4, MathSAT5, SMT Interpol, Yices2, Z3
  - Kind2 : CVC4, Yices2, Z3

Model checking : régulateur de vitesse (suite)

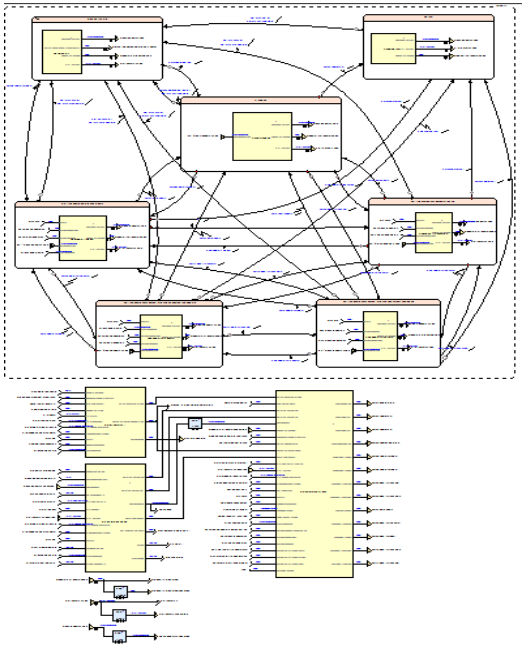
- Trois modèles à vérifier
  - M1 : Sous-modèle gérant l'autorisation de la fonction => 300 lignes de code Lustre
  - M2 : Modèle RVV (linéaire, entiers) => 1300 lignes de code Lustre
  - M3 : Modèle RVV évolué (non-linéaire, flottants) => 2000 lignes de code Lustre



M1 : Sous-modèle gérant l'autorisation de la fonction



M2 : Modèle RVV (linéaire, entiers)

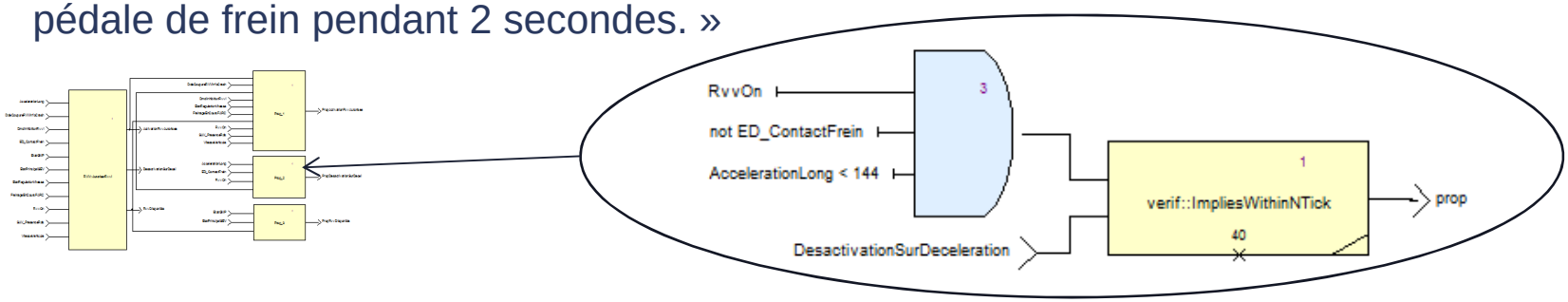


M3 : Modèle RVV évolué (non-linéaire, flottants)

© PSA Group. All rights reserved.

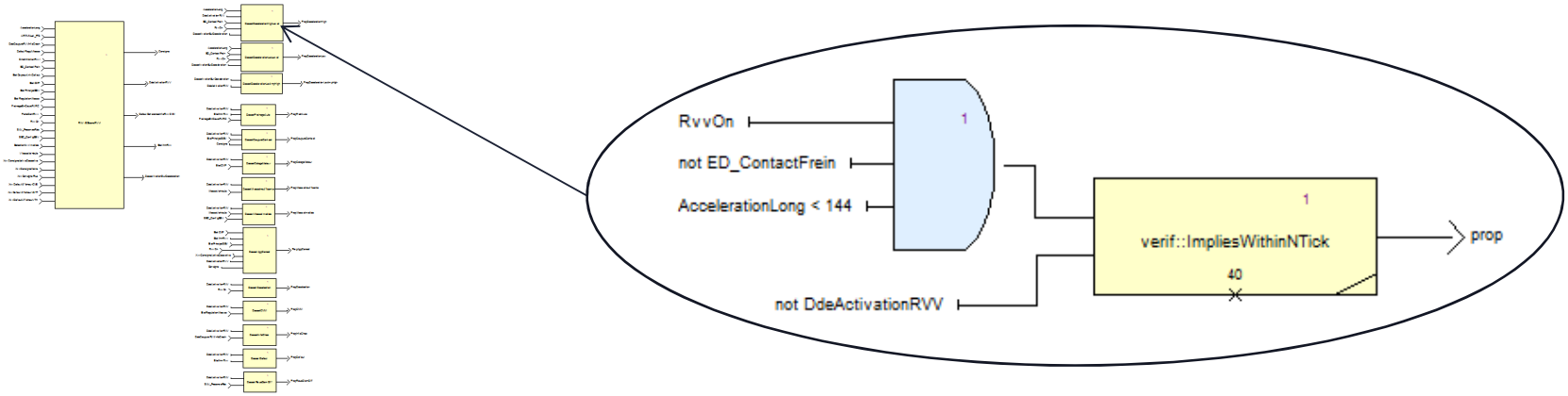
Model checking : zoom sur les propriétés

- P1 sur M1 : « Désactivation du régulateur sur décélération véhicule sans appui sur la pédale de frein pendant 2 secondes. »



- P2_Low sur M2 : Même propriété que P1 aux bornes de la fonction « Autorisation » => nécessité de tirer des fils pour les remonter aux bornes du système

- P2_High sur M2 : Même propriété que P1 mais aux bornes du système



© PSA Group. All rights reserved.

Model checking : résultats

- Synthèse model checkers :

Model checker	Solveur SMT	*k*-induction	Invgen	PDR/IC3
Design Verifier	Prover	Oui	Oui	Non
GATeL	Colibri	Oui	Oui	Non
JKind	CVC4	Oui	Oui	Oui
JKind	MathSAT5	Oui	Oui	Oui
JKind	SMT Interpol	Oui	Oui	Oui
JKind	Yices2	Oui	Oui	Oui
JKind	Z3	Oui	Oui	Oui
Kind2	CVC4	Oui	Oui	Oui
Kind2	Yices2	Oui	Oui	Non
Kind2	Z3	Oui	Oui	Oui

- Synthèse résultats :
  - La *k*-induction suffit globalement pour des propriétés de bas niveau sur des blocs de taille petite et avec un nombre de pas de calcul faible
  - La génération d'invariants (invgen) aide la *k*-induction à finir plus vite la preuve de validité mais peut parfois ralentir le calcul si trop d'invariants sont générés
  - Certains model checkers vérifient toutes les propriétés en parallèle avec la *k*-induction : P2_High prend beaucoup plus de temps à être résolue toute seule que si on demande la P2_High et la P2_Low en même temps

© PSA Group. All rights reserved.

Model checking : résultats (suite) & conclusion

Synthèse résultats :

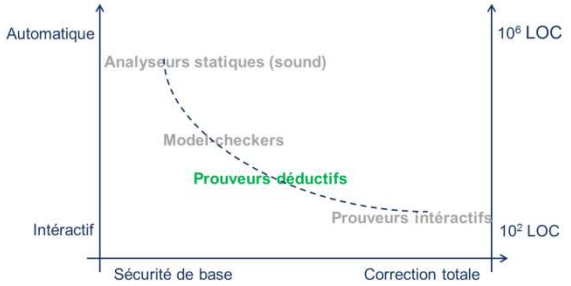
- PDR/IC3 est une technique de model checking plus efficace pour les propriétés affaiblies mais aussi pour des modèles de taille importante. Il est utilisé parfois pour la génération d'invariants qui aide à résoudre des propriétés par *k*-induction.
- L'augmentation du nombre de pas pose généralement problème
- On ne constate pas d'influence dans l'ordre dans lequel sont données les propriétés
- Résultats très variables en fonction du model checker et du solveur SMT utilisé

Conclusion

- Détection de bugs subtils non vus en test
- Automatique et fournissant des contre-exemples
- Exploitation des contre-exemples : besoin d'un simulateur
- Bon découpage d'architecture pour un meilleur passage à l'échelle
- Abstraction par contrats et raisonnement compositionnel pour les parties non linéaires
- Options et logs pas toujours disponibles
- Besoin de guidage de l'utilisateur pour pouvoir utiliser le bon model checker au bon endroit

© PSA Group. All rights reserved.

Preuve déductive : mise en œuvre



- La fonction RVVi fait appel à une fonction qui calcule la racine carrée de 0 à 100,00 par interpolation linéaire

- Objectif : démontrer que le calcul de racine carrée par interpolation linéaire reste borné et proche de la valeur réelle

- Spécification :
  - HLR : Besoin de calcul de racine carrée
  - LLR : Propose une méthode de calcul de la racine carrée avec des entiers, approximée par interpolation linéaire avec 41 valeurs

- Démarche :
  - D'abord prouver le bon calcul pour les 8 premières valeurs interpolées
  - Ensuite prouver le bon calcul pour les 41 valeurs

- Outils : Frama-C, SPARK

Frama-C : preuve de calcul de racine carrée interpolée

- Preuve avec 8 valeurs (Frama-C Silicon-20161101) :

```
$ frama-c -wp -wp-rte sqrt3.c
[wp] 87 goals scheduled
[wp] Proved goals:   87 / 87
     Qed:            63  (4ms-5ms-16ms)
     Alt-Ergo:       24  (8ms-249ms-4.3s) (117)
```

- Preuve avec 41 valeurs (cas réel) ne passe pas à l'échelle (Frama-C Silicon-20161101) :

```
$ frama-c -wp -wp-rte -wp-timeout 300 sqrt4.c
[wp] 222 goals scheduled
[wp] [Alt-Ergo] Goal typed_InterpolLineaire_post : Unknown (Qed:12ms) (2.8s)
[wp] [Alt-Ergo] Goal typed_InterpolLineaire_assert_3 : Timeout (Qed:4ms) (5')
[wp] Proved goals:  220 / 222
     Qed:           195  (4ms-10ms-136ms)
     Alt-Ergo:       25  (12ms-7.2s-2'7s) (117) (interrupted: 1) (unknown: 1)
```


© PSA Group. All rights reserved.

Frama-C : preuve de calcul de racine carrée interpolée (suite)

- Contrat non prouvé :

```
/*@ requires 0 <= Xa <= 10000 && 0 <= Xb <= 10000;
  @ requires 0 <= Ya <= 1000 && 0 <= Yb <= 1000;
  @ requires Yb > Ya && Xb >= Xa;
  @ requires Xa <= X <= Xb;
  @ ensures Xa != Xb ==> \result == (Ya + (X - Xa) * (Yb - Ya) / (Xb - Xa));
  @ ensures Xa == Xb ==> \result == Ya;
  @ assigns \nothing;
  @*/
```

- Explication : difficulté pour tous les solveurs (Alt-Ergo, CVC4, E-prover, Yices, Z3) de démontrer qu'il ne comporte pas d'overflow
- Solution proposé par le CEA : utiliser le solveur Colibri qui prend en compte l'arithmétique entière modulaire avec la version Frama-C Phosphorus-20170501+dev :

```
$ frama-c -wp -wp-rte -wp-prover native:colibri sqrt4.c
[wp] 54 goals scheduled
[wp] Proved goals:   54 / 54
     Qed:               27  (4ms-32ms-280ms)
     Colibri (native):  27  (552ms-1.2s-9.8s)
```


© PSA Group. All rights reserved.

SPARK : preuve de calcul de racine carrée interpolée

- Idée : comparer SPARK à Frama-C pour la fonction complète avec 41 valeurs
- Résultat : la preuve est finie en moins de 100 s en utilisant CVC4 et Z3
- Astuce : utilisation de bitvectors pour les entiers

```
type Unsigned is mod 2**32;
subtype TWORD is Unsigned range 0 .. 65535;
subtype TBYTE is Unsigned range 0 .. 255;
```

- Avantage : contre-exemples

```
sqrt3_ko.ads:32:35: medium: contract case might fail (e.g. when ApproximationRacine'Result = 5 and
ContenuRacine = 5)
```

- Test dynamique pour vérifier le contrat :

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE : failed contract case at sqrt3_ko.ads:34
```

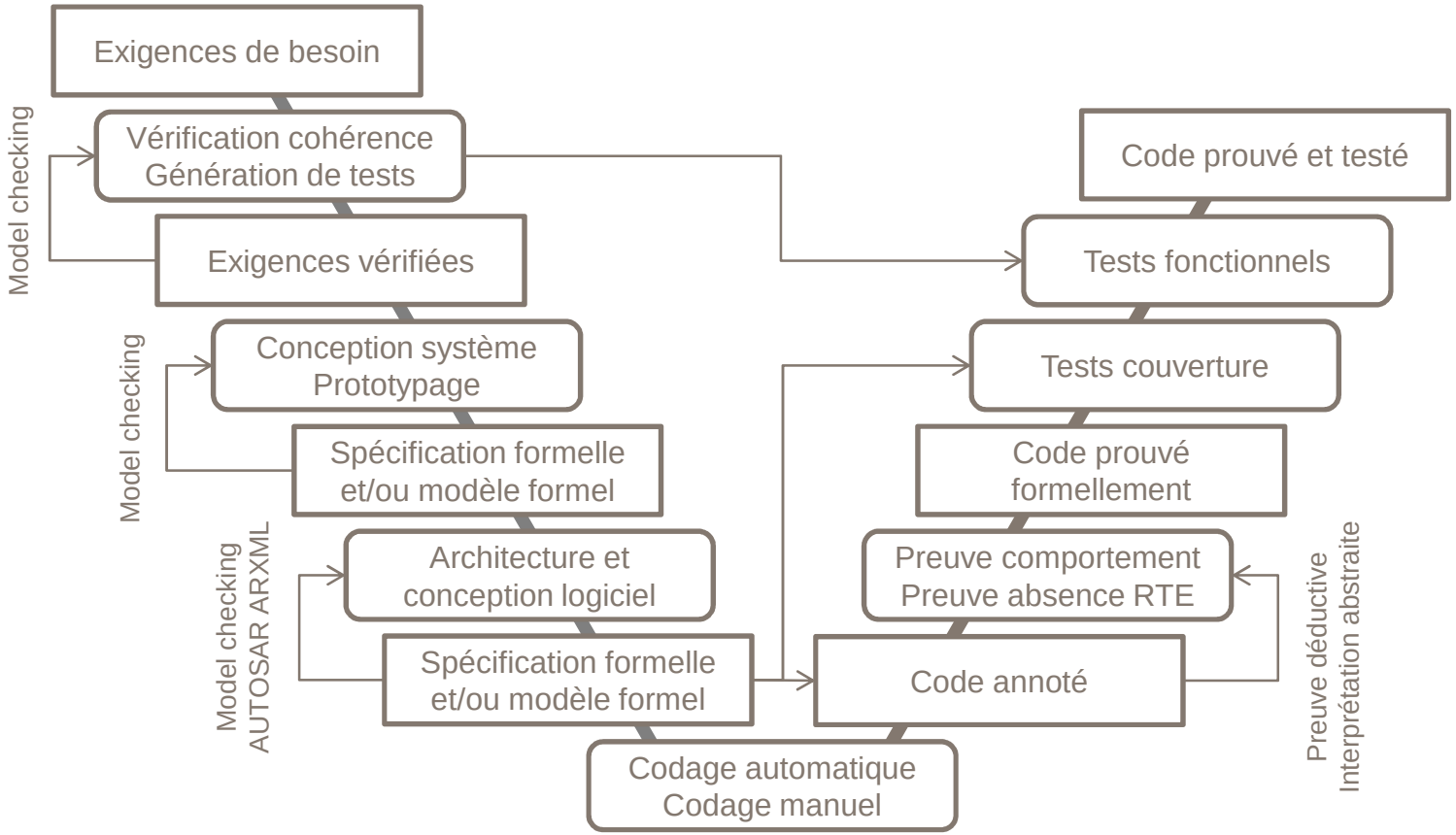

© PSA Group. All rights reserved.

3

VERS UN PROCESSUS UTILISANT DES MÉTHODES FORMELLES

Vers un processus utilisant des méthodes formelles

- Introduction progressive de méthodes formels sur des modules ASIL >= B
- Identification d'exigences fonctionnelles critiques à prouver formellement
- Architecture logicielle facilitant l'usage des méthodes formelles



© PSA Group. All rights reserved.

4

CONCLUSION

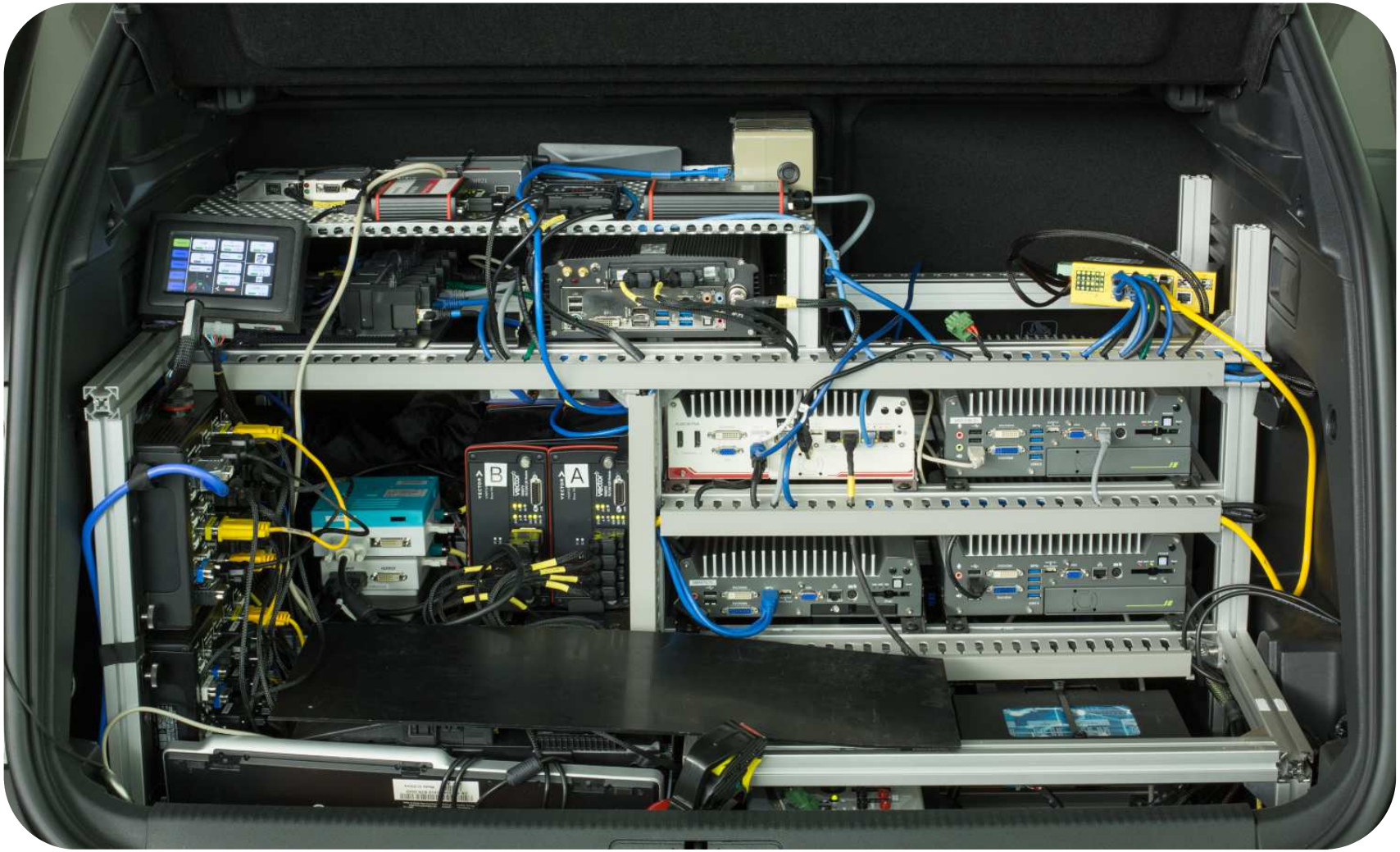
Conclusion

- Les méthodes formelles ont permis
  - Un gain en confiance : détection de bugs non vus par les tests
  - Une analyse de la spécification sous un autre angle permettant de la rendre plus robuste et cohérente
- Les méthodes formelles nécessitent des outils
  - Complexes qui peuvent comporter des bugs. Nécessité d'utiliser plusieurs outils pour augmenter la confiance
  - Utilisant des heuristiques efficaces pour certains types de problèmes mais pas de manière générale
  - Parfois développés par des étudiants ou post-doctorants plus maintenus après la fin du projet
  - Commercialisés pas forcément bien documentés sur les aspects techniques des algorithmes utilisés
- Quelques défis à relever
  - Passage à l'échelle
  - Arithmétique non linéaire, flottants
  - Tables de correspondance (lookup tables), compteurs, timers, intégrateurs

"Formal methods will never have a significant impact until they can be used by people who don't understand them"
Tom Melham

© PSA Group. All rights reserved.

Merci pour votre attention. Questions ?



© PSA Group. All rights reserved.

Bibliographie

- Vérification spécification
  - Aaron W. Fifarek, Lucas G. Wagner, Erika R. Hoffman, Benjamin D. Rodes, M. Anthony Aiello, and Jennifer A. Davis. “SpeAR v2.0: Formalized Past LTL Specification and Analysis of Requirements” 2017. http://loonwerks.com/publications/pdf/wagner2017nfm_spear.pdf
  - « Mise au point et test d'exigences temps-réel avec STIMULUS » Erwan Jahier (VERIMAG) - Bertrand Jeannet (ARGOSIM), FMF5 : http://projects.laas.fr/IFSE/FMF/J5/slides/P05_Bertrand_Jeannet.pdf
- *k*-induction
  - Principe: http://www.ccs.neu.edu/home/wahl/Publications/k-induction.pdf
  - Donaldson, Alastair F., Leopold Haller, Daniel Kroening, and Philipp Rümmer. “Software Verification Using K-Induction.” In *Proceedings of the 18th International Conference on Static Analysis*, 351–68. SAS’11. Berlin, Heidelberg: Springer-Verlag, 2011. http://www.kroening.com/papers/sas2011.pdf
- Property Directed Reachability / IC3
  - Bradley, Aaron R., and Zohar Manna. “Property-Directed Incremental Invariant Generation.” *Formal Aspects of Computing* 20, no. 4 (2008): 379–405. doi:10.1007/s00165-008-0080-9. http://theory.stanford.edu/~arbrad/papers/pdiig.pdf
  - Bradley, Aaron R. “Understanding IC3.” In *Theory and Applications of Satisfiability Testing – SAT 2012: 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, edited by Alessandro Cimatti and Roberto Sebastiani, 1–14. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. http://theory.stanford.edu/~arbrad/papers/Understanding_IC3.pdf
- Model checking utilisant SAT/SMT
  - Bouali, Amar, and Bernard Dion. “Formal Verification for Model-Based Development.” In *SAE Technical Paper 2005-01-0781*, 2005. https://www.researchgate.net/publication/237535420_Formal_Verification_for_Model-Based_Development
  - Hagen, George Edward, and Cesare Tinelli. “Scaling Up the Formal Verification of Lustre Programs with SMT-Based Techniques.” In *2008 Formal Methods in Computer-Aided Design*, 1–9, 2008. http://clc.cs.uiowa.edu/Kind/Papers/FMCAD-08.pdf
  - Champion, Adrien, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. “The Kind 2 Model Checker.” In *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, edited by Swarat Chaudhuri and Azadeh Farzan, 510–17. Cham: Springer International Publishing, 2016. http://www.sticksel.info/christoph/files/CAV2016-Kind2.pdf
  - Jovanović, Dejan, and Bruno Dutertre. “Property-Directed K-Induction.” In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*, 85–92. FMCAD ’16. Austin, TX: FMCAD Inc, 2016. http://csl.sri.com/users/dejan/papers/jovanovic-fmcad2016.pdf
- Interprétation abstraite
  - Cousot, Patrick, and Radhia Cousot. “A Gentle Introduction to Formal Verification of Computer Systems by Abstract Interpretation.” In *Logics and Languages for Reliability and Security*, edited by J. Esparza, O. Grumberg, and M. Broy, 1–29. NATO Science Series III: Computer and Systems Sciences. IOS Press, 2010. https://hal.inria.fr/inria-00543886
- Preuve déductive
  - Introduction to Ada & SPARK, Michigan State University, 2014. https://www.cse.msu.edu/~cse814/Lectures/09_spark_intro.pdf
  - Kirchner, Florent, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. “Frama-C: A Software Analysis Perspective.” *Formal Aspects of Computing* 27, no. 3 (2015): 573–609. http://julien.signoles.free.fr/publis/2015_fac.pdf

© PSA Group. All rights reserved.