Formal Verification of Software

Αλέξανδρος Πετρίδης, Αλέξανδρος Οικονόμου

I. Εισαγωγή

Η συνολική δομή αυτής της εργασίας έχει 6 ενότητες, συμπεριλαμβανομένου και της εισαγωγής. Η ενότητα II παρουσιάζει μια ανάλυση στην έννοια του Formal Veriifcation, τους λόγους που αυτό πρέπει να χρησιμοποιείται -αναφέροντας κάποια παραδείγματα- και το ταξινομεί σε δύο μεγάλες κατηγορίες, αναλύοντας τα οφέλη και τους περιορισμούς της κάθε μιας. Η ενότητα III μας αναλύει τη μέθοδο πιστοποίησης Event-B και δίνει ένα παράδειγμα υλοποίησης της σε λειτουργικά συστήματα IoT . Η ενότητα IV περιγράφει το πρόβλημα που υπάρχει στην πιστοποίηση αυτόνομων οχημάτων. Η ενότητα V αναφέρει τις μεθοδολογίες της πιστοποίησης αυτής. Τέλος, η ενότητα VI αναλύει την αρχιτεκτονική που υπάρχει στην πιστοποίηση του λογισμικού.

Σχήμα 1. 66 MHz Intel Pentium με το FDIV bug.

II. Ανάλυση του Formal Verification

Οι τεχνικές μηχανικής λογισμικού που χρησιμοποιούνται σήμερα μας επιτρέπουν να δημιουργήσουμε συστήματα μεγάλης πολυπλοκότητας. Ωστόσο, οι τεχνικές που χρησιμοποιούνται για την επαλήθευση αυτών δεν έχουν ακόμα την ίδια δυναμική. Ακόμα και σχετικά απλά συστήματα μπορεί να έχουν προβλήματα τα οποία δεν μπορούν εύκολα να εντοπιστούν και να αποκαλυφθούν μέσω της προσομοίωσης ή των δοκιμών. Ένα παράδειγμα είναι το floating point bug στον επεξεργαστή Pentium της Intel , το οποίο ήταν αρκετά σπάνιο ώστε να μην αποκαλυφθεί μέσω εκτεταμένων δοκιμών, αλλά όχι τόσο σπάνιο ώστε να αποτραπεί η ανακάλυψή του από τους χρήστες μέσα σε ένα χρόνο λειτουργίας του.

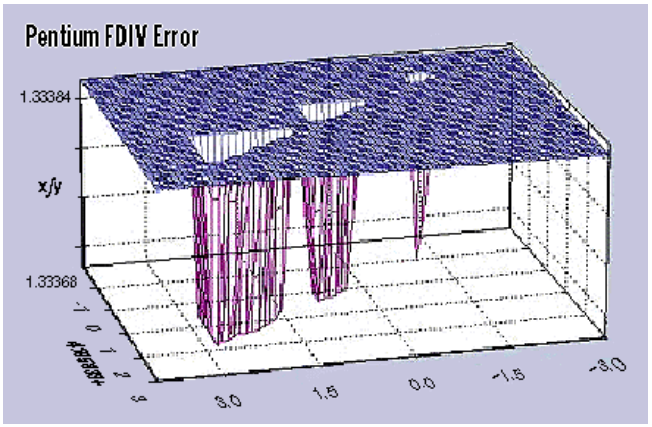
Το σφάλμα αφορούσε το FPU (Floating-Point Unit)σε έναν από τους πρώτους Intel Pentium επεξεργαστές. Λόγω του σφάλματος, ο επεξεργαστής επέστρεφε εσφαλμένα αποτελέσματα δυαδικής κινητής υποδιαστολής κατά την διαίρεση ορισμένων ζευγών αριθμών υψηλής ακρίβειας. Το σφάλμα ανακαλύφθηκε το 1994 από έναν καθηγητή μαθηματικών στο Lynchburg College, ονόματι Thomas R.Nicely. Ενώ αυτά τα σφάλματα στις περισσότερες περιπτώσεις εμφανίζονται σπάνια και έχουν ως αποτέλεσμα μικρές αποκλίσεις από τις σωστές τιμές εξόδου, σε ορισμένες περιπτώσεις συνέβαιναν συχνά και οδηγούσαν σε πιο σημαντικές αποκλίσεις.

Για να γίνει testing σε μια απλή διαίρεση με floating points χρειάζεται 2^{128} διαφορετικά τεστ, κάτι το οποίο δεν είναι πάντα εφικτό. Για την ιστορία αυτό το σφάλμα κόστισε στην εταιρία περίπου 475 εκατομμύρια δολάρια.

Ένα ακόμα παράδειγμα υπέρ του επιχειρήματος των Formal Verification εργαλείων είναι το λάθος μίας γραμμής το οποίο προκάλεσε την κατάρρευσή του νεοσύστατου κρυπτονομίσματος YAM μόλις δύο μέρες μετά την κυκλοφορία του. Η παρακάτω γραμμή κόστισε περίπου μισό εκατομμύριο δολάρια.

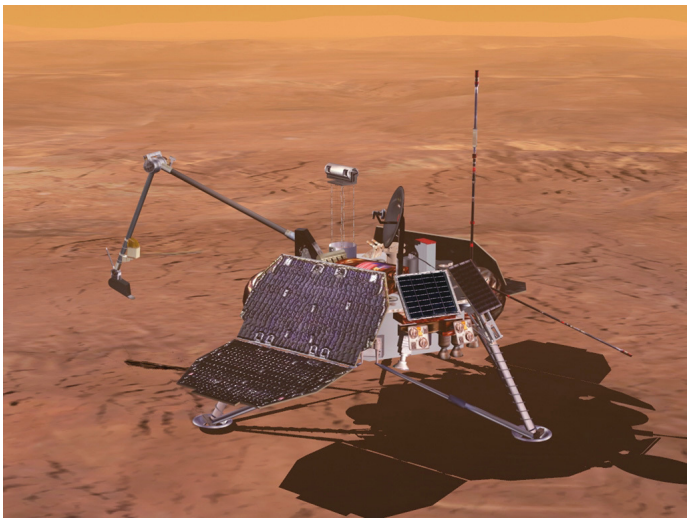
```
totalSupply = initSupply.mul(yamsScalingFactor);
```

Όταν η σωστή γραμμή θα έμοιαζε κάπως έτσι.



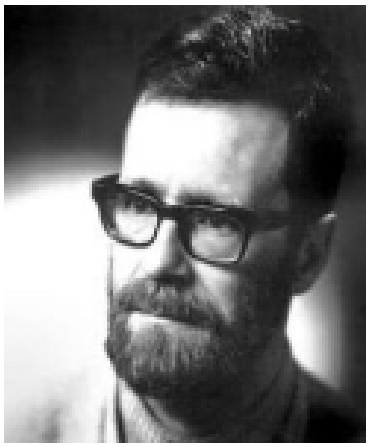
Σχήμα 2. Τρισδιάστατο γράφημα που δείχνει ένα παράδειγμα σφάλματος του Intel Pentium . Στην διαίρεση $\frac{4195835}{3145727}$ τα συμπιεσμένα προς τα κάτω τρίγωνα δείχνουν τις περιοχές που έχουν υπολογιστεί λανθασμένες τιμές. Η σωστές τιμές στρογγυλοποιούνται στο 1.3338, ενώ οι τιμές που επέστρεφε ο Intel Pentium στο 1.3337, ένα λάθος στο 5^o πιο σημαντικό bit

```
totalSupply = initSupply.mul(yamsScalingFactor).div(BASE);
```



Σχήμα 3. Το Mars Polar Lander μια φιλόδοξη αποστολή στον Άρη χάθηκε κατά την άφιξή του στις 3 Δεκεμβρίου 1999 λόγω ενός ¨ανόητου' σφάλματος, όπως το χαρακτήρισαν οι ειδικοί, στο λογισμικό.

Τα συστήματα λογισμικού έχουν περιτριγυρίσει την καθημερινή μας ζωή. Εκτός από τους υπολογιστές και τα κινητά τηλέφωνα, υπάρχουν συστήματα τα οποία χειρίζονται κρίσιμα συστήματα, όπως τα αεροπλάνα, τα αυτοκίνητα και οι ελεγκτές στην βιομηχανία. Τα σφάλματα σε τέτοια περιβάλλοντα μπορεί να οδηγήσουν σε σοβαρούς τραυματισμούς και τεράστιες οικονομικές ζημιές. Επομένως, εκτός από τις παραδοσιακές προσεγγίσεις επαλήθευσης όπως το **software testing**, σε αυτούς τους τομείς συνήθως χρησιμοποιούνται επίσημες τεχνικές επαλήθευσης για να αυξήσουν την ποιότητα και την αξιοπιστία του συστήματος.



Σχήμα 4. Ο Edgen Dijkstra δήλωσε πως η διδασκαλία της αποδοτικότητας των formal μεθόδων είναι μια από τις χαρές τις ζωής.

Α΄. *Formal Verification*

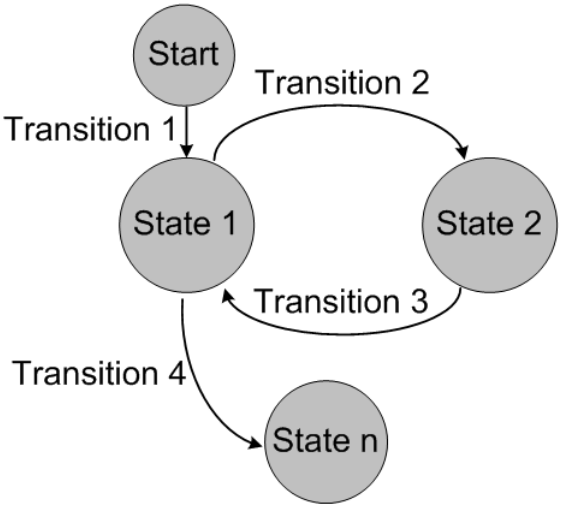
Το **Formal Verification** του κώδικα είναι μια τεχνική, η οποία μπορεί να εγγυηθεί την απουσία σφαλμάτων. Ακόμα, δείχνει συνέπεια μεταξύ δύο διαφορετικών περιγραφών ενός προγράμματος. Συχνά, η μία περιγραφή είναι ο ίδιος ο κώδικας του προγράμματος και η άλλη οι προδιαγραφές του. Το **Formal Verification** του κώδικα αντιμετωπίζει τις προδιαγραφές και τα προγράμματα ως επίσημα, μαθηματικά κείμενα και η επίδειξη συνέπειας μεταξύ τους παίρνει τη μορφή μαθηματικής απόδειξης. Το **Formal Verification** του κώδικα θα πρέπει να θεωρείται σημαντικό στοιχείο για την απόδειξη αξιοπιστίας για κρίσιμα συστήματα και ακόμη πιο σημαντικό στοιχείο στη διαδικασία κατασκευής τέτοιων συστημάτων. Όταν ενσωματώνεται στη διαδικασία σχεδιασμού και ανάπτυξης (αντί να είναι μια εκ των υστέρων ανάλυση), το **formal verification** μπορεί να συμβάλει σημαντικά στην ανάπτυξη αξιόπιστων συστημάτων ενθαρρύνοντας τη συστηματική και συνειδητή σκέψη και την προτίμηση για απλό και ξεκάθαρο σχεδιασμό και υλοποίηση.

Τα επίσημα εργαλεία επαλήθευσης μπορούν να παρέχουν την εγγύηση ότι ένα σύστημα δεν έχει συγκεκριμένα ελαττώματα. Γενικά, τα εργαλεία **formal** μεθόδων χωρίζονται σε δύο κατηγορίες με βάση τον **J.Lou** [1] και την ομάδα του :

Σχήμα 5. Ο Bertand Meyer θεωρεί το formal verification επανάσταση, και αναφέρει πως δεν ξέρει πότε αλλά εν τέλη θα συμβεί η αναμενόμενη εξέλιξη της χρήσης του καθώς η θεωρία του είναι έτοιμη.

1) Model Checkers, εργαλεία τα οποία ελέγχουν τον σχεδιασμό σε σχέση με τις καθορισμένες ιδιότητες. Το πλήθος τους θα ελέγξει αυτόματα με περιορισμένη ανθρώπινη πρόσβαση το μοντέλο και θα επιστρέψει ένα από τα παρακάτω αποτελέσματα:
   α′) Οι ιδιότητες ικανοποιούν τον σχεδιασμό.
   β′) Δεν ικανοποιούνται οι ιδιότητες, με παροχή αντιπαραδειγμάτων.
   γ′) Απροσδιόριστη κατάσταση, καθώς ο χώρος κατάστασης είναι τέτοιος που το εργαλείο δεν μπορεί να υπολογίσει ένα αποτέλεσμα σε εύλογο χρονικό διάστημα.
2) Theorem Provers, εργαλεία τα οποία ονομάζονται και proof assistants συνδυάζουν αυτοματοποιημένες τεχνικές με χειροκίνητη καθοδήγηση για την απόδειξη της ορθότητας. Είναι γενικά πιο ισχυρά εργαλεία από τα Model Checkers .

Γενικά, τα model checking εργαλεία ενσωματώνουν αυτό που μπορεί να γίνει εντελώς αυτόματα και τα theory proving εργαλεία παρέχουν ένα μέσο το οποίο χρειάζεται ανθρώπινη παρέμβαση και δημιουργικότητα. Τέλος, τα model checking εργαλεία έχουν πολλά οφέλη προς τους προγραμματιστές με μικρή εξειδίκευση στις formal μεθόδους οι οποίοι χρειάζονται απλώς ένα εργαλείο, όμως μπορεί να είναι περιορισμένα σε αυτά που μπορούν να αποδείξουν. Τα theory proving εργαλεία από την άλλη μεριά είναι περισσότερο ικανά αλλά απαιτούν περισσότερη τεχνογνωσία και μπορεί να είναι δύσκολη η χρήση



Σχήμα 6. Παράδειγμα finite-state machine τα οποία χρησιμοποιούνται στην μοντελοποίηση ενός συστήματος για τα εργαλεία model-checking

τους για επαναλαμβανόμενες εργασίες.

Β′. Περιορισμοί του *Theory Prooving*

Τα Theory Prooving εργαλεία χρειάζονται αρκετή εξειδίκευση ανθρώπων για την απόδειξη μικρών θεωρημάτων, καθώς είναι αρκετά δύσκολο να αποδειχθούν μεγάλα ή σκληρά θεωρήματα με τα συγκεκριμένα εργαλεία. Ακόμα όπως προείπώθηκε χειρίζονται μόνο από ειδικούς, καθώς χρειάζονται βαθιά κατανόηση τόσο του σχεδιασμού του συστήματος, όσο και της μεθοδολογίας απόδειξης ορθότητάς του. Τέλος μπορούν να αυτοματοποιηθούν μονάχα για πολύ λίγες κατηγορίες σχεδιασμού προγραμμάτων.

Σχήμα 7. Ο Mike Holloway, μηχανικός της NASA αναφέρει πως οι formal μέθοδοι είναι τα μαθηματικά των software engineers, δείχνοντας έτσι την χρησιμότητα τους.

Γ΄. Δυνατότητες του *Model Checking*

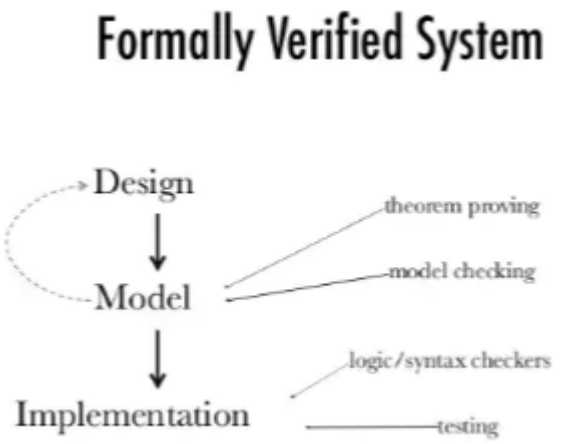
Το Model Checking μπορεί να δώσει γενική πιστοποίηση η οποία είναι εφαρμόσιμη σε μια πληθώρα συστημάτων. Υποστηρίζει μερική πιστοποίηση, καθώς οι ιδιότητες μπορούν να ελέγχουν ξεχωριστά με προτεραιότητα στις απαραίτητες για την λειτουργία του συστήματος. Ακόμα παρέχει διαγνωστικές πληροφορίες σε κάθε περίπτωση που η ιδιότητα είναι άκυρη, η πληροφορία αυτή είναι πολύ χρήσιμη στους προγραμματιστές όταν κάνουν debugging. Επιπροσθέτως, σε αντίθεση με το Theory Prooving δεν χρειάζεται ούτε μεγάλη αλληλεπίδραση με τον χρήστη, ούτε μεγάλη εμπειρία του χρήστη για να το χειριστεί. Τέλος, μπορεί εύκολα να ενσωματωθεί στα ήδη υπάρχοντα εργαλεία ανάπτυξης λογισμικού και έχει ισχυρό μαθηματικό υπόβαθρο βασισμένο στην θεωρία αλγορίθμων γραφημάτων, τις δομές δεδομένων, την λογική, κ.α.

Δ΄. Περιορισμού του *Model Checking*

Το Model Checking χρησιμοποιείται κυρίως σε συστήματα που έχουν να ασχοληθούν με τον έλεγχο κάποιου συστήματος, παρά σε συστήματα που έχουν να κάνουν με την επεξεργασία δεδομένων. Επίσης το Model Checking πιστοποιεί ένα μοντέλο και όχι το πραγματικό σύστημα, γι΄ αυτόν τον λόγο χρειάζονται και συμπληρωματικές τεχνικές όπως είναι το testing . Ακόμα, ελέγχει μόνο τις ιδιότητες που έχουν δηλωθεί, άρα εξ΄ ορισμού του δεν εγγυάται την ορθότητα του συστήματος. Επιπροσθέτως, υποφέρει από το πρόβλημα στατε εξπλοσιον, δηλαδή την ταχεία αύξηση της πολυπλοκότητας ενός προβλήματος λόγω του τρόπου με τον οποίο η συνδυαστική του προβλήματος επηρεάζεται από την είσοδο, τους περιορισμούς και τα όρια του προβλήματος και χρειάζεται κάποια εμπειρία στο να βρεθούν τα κατάλληλα επίπεδα για να αποκτηθούν μικρότερα μοντέλα συστημάτων και να δηλωθούν ιδιότητες στον λογικό φορμαλισμό που χρησιμοποιείται. Τέλος, δεν επιτρέπει γενικούς ελέγχους και ένα λογισμικό ενός εργαλείου Model Checking μπορεί να περιέχει και αφ΄ εαυτού του κάποια ελαττώματα λογισμικού.

III. Μέθοδος πιστοποίησης: Event-B

Η Event-B είναι μια formal μέθοδος για μοντελοποίηση και ανάλυση συστημάτων. Η υποχρέωση απόδειξης χαρακτηρίζει την σημασιολογία του μοντέλου Event-B. Το Event-B εστιάζει στην υποχρέωση απόδειξης: εκφράζει το σύστημα σε διαφορετικά επίπεδα-μοντέλα και χρησιμοποιεί μαθηματικές αποδείξεις για να διασφαλίζει την συνοχή τους. Το Event-B είναι μια Formal Verification τεχνική η οποία λειτουργεί με events (γεγονότα).

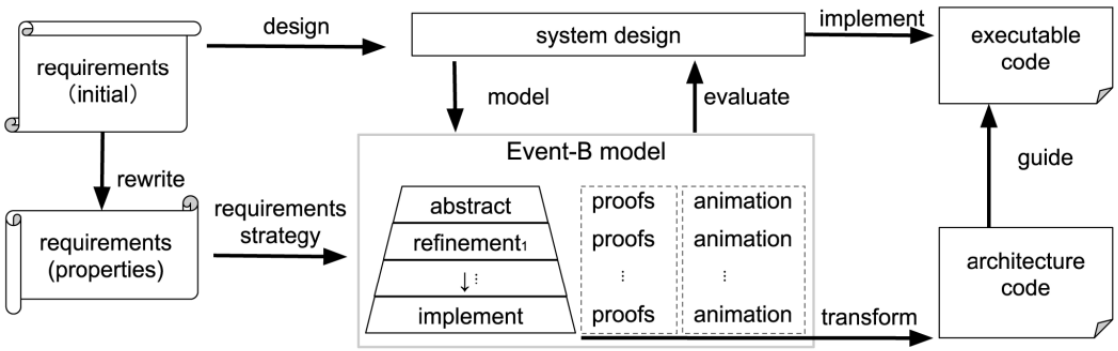


Σχήμα 8. Μορφή επίσημα επαληθευμένου συστήματος

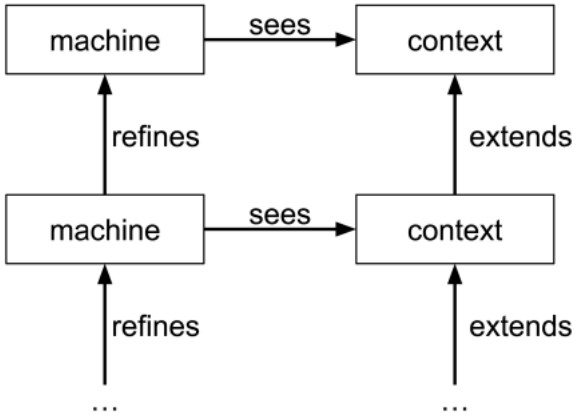
Η Event-B μέθοδος αποτελείται από δύο μέρη: το δυναμικό και το στατικό. Το στατικό μέρος του μοντέλου περιγράφεται στο αρχείο συμφραζομένων, το οποίο περιέχει το σύνολο των διανυσμάτων, τις σταθερές, τα αξιώματα, κτλ. Το δυναμικό μέρος του μοντέλου υλοποιείται από τις μηχανές και περιέχει μεταβλητές, αμετάβλητες σταθερές, θεωρήματα, γεγονότα, κλπ. Με την σχέση που τα διακατέχει να συνοψίζεται στο εξής: οι μηχανές βλέπουν τα συμφραζόμενα, τα συμφραζόμενα μπορούν να επεκταθούν, οι μηχανές μπορούν να βελτιωθούν. Η σχέση ανάμεσα στην μηχανή και τα συμφραζόμενα στο Event-B φαίνεται στο σχήμα 10. Αυτή η μέθοδος βασίζεται στις έννοιες των προϋποθέσεων και των μετα-υποθέσεων της Hoare λογικής, της πιο αδύναμης προϋπόθεσης του Dijkstra και του λογισμού αντικατάστασης. Η Hoare λογική είναι ένα formal σύστημα λογικών κανόνων για τον αυστηρό συλλογισμό σχετικά με την ορθότητα των προγραμμάτων των υπολογιστών.Ένα γεγονός μπορεί να πάρει την παρακάτω μορφή:

event $\triangleq$ **any** $\times$ **where** G **then** Act **end**.

Όπου x είναι η λίστα με τις παραμέτρους, G αντιπροσωπεύει τους περιορισμούς που θέτουμε σε



Σχήμα 9. Δομή Formal Verification με χρήση σε λειτουργικά συστήματα IoT βασισμένο στην Event-B μέθοδο.



Σχήμα 10. Η σχέση μεταξύ της abstract μηχανής και του αρχείου συμφραζομένων στην Event-B μέθοδο.

ένα event (γεγονός) και Act είναι η πράξη που τροποποιεί ορισμένες από τις μεταβλητές των καταστάσεων. Όταν ο περιορισμός ικανοποιείται, το γεγονός δεν είναι πλέον χρήσιμο.

Α΄. *Formal Verification* σε *IoT* λειτουργικά συστήματα

Η δομή που φαίνεται στο σχήμα 15 χρησιμοποιείται για την πιστοποίηση λειτουργικών συστημάτων σε IoT περιβάλλοντα [2]. Σε αυτή την περίπτωση οι προδιαγραφές ξαναγράφονται για να γίνει πιο ακριβείς η περιγραφή τους. Οι επαναπροσδιορισμένες προδιαγραφές λαμβάνονται προσεκτικά υπόψη και συνοψίζονται σε μια γραφή από πάνω προς τα κάτω όπου όσο κατεβαίνουνε έχουμε τις βελτιωμένες προδιαγραφές οι οποίες οδηγούν στην υλοποίηση. Οι απαιτήσεις που έχουν ξαναγραφτεί λαμβάνονται προσεκτικά υπόψη και συνοψίζονται σε μια top-down κατακόρυφη λίστα βελτιστοποίησης. Μια αντιστοίχηση μεταξύ απαιτήσεων και της βελτίωσης έχει σχεδιαστεί για να αποκτηθεί μια στρατηγική βελτίωσης και να επιβεβαιωθεί ότι αυτή η στρατηγική βελτίωσης είναι ικανοποιητική. Σύμφωνα με τη στρατηγική βελτίωσης, οι απαιτήσεις μετατρέπονται σε γεγονότα και ενέργειες για την ολοκλήρωση της κάθετης βελτίωσης του μοντέλου.

Προσθέτονται χαρακτηριστικά ή τελειοποιούνται λειτουργίες μέσω μιας σειράς βελτιώσεων του μοντέλου, ολοκληρώνεται το μοντέλο βήμα προς βήμα και τέλος προσομοιώνεται ολόκληρο το σύστημα. Οι σταθερές και τα θεωρήματα που περιγράφουν τις ιδιότητες στο μοντέλο μπορούν να μετατραπούν σε αντίστοιχες υποχρεώσεις απόδειξης βασισμένες στους κανόνες. Η ορθότητα των ιδιοτήτων του μοντέλου είναι εγγυημένη με την πλήρη απόδειξη όλων των σχετικών υποχρεώσεων απόδειξης που έχουν παραπάνω δημιουργηθεί. Οι ιδιότητες που έχουν ήδη αποδειχθεί ότι ισχύουν στα πρώτα μοντέλα είναι εγγυημένο ότι θα διατηρηθούν και στα μεταγενέστερα. Αυτή η δομή είναι κατάλληλη για την επαλήθευση ενός συστήματος στη φάση σχεδιασμού και το αποτέλεσμα της είναι μια αξιολόγηση της ασφάλειάς του. Επιπλέον το μοντέλο Event-B μπορεί να μετατραπεί στην αρχιτεκτονική του κώδικα η οποία μπορεί έπειτα να καθοδηγήσει σωστά την σύνταξη του εκτελέσιμου κώδικα του συστήματος.

Notation	Meaning
$\exists$	There exists
$\forall$	For all
$\in$	Element of
$\notin$	Not an element of
$\vee$	Logical disjunction
$\wedge$	Logical conjunction
$\Rightarrow$	Logical implication
$\mapsto$	Maplet
$\rightarrow$	Total function
$\leftrightarrow$	Relation
$\mathbb{P}$	The set of all subsets (power set)
$\mathbb{N}_1$	The set of positive natural numbers
$:=$	Becomes equal to
$\varnothing$	Empty set
$\cup$	Union
$\bigcup$	N-ary union
$\setminus$	Set minus
$=$	Equal to
$\neq$	Not equal to
$..$	Upto sign
$\rhd$	Range restriction
$\mid$	Such that

Σχήμα 11. Μαθηματικοί τελεστές που χρησιμοποιούνται στην Event-B μέθοδο.

IV. Αυτόνομα Αυτοκίνητα

Αυτόνομα αυτοκίνητα παρουσιάζονται ως λύση σε αρκετά προβλήματα που προκαλούνται κυρίως από τον άνθρωπο στους δρόμους, όπως τροχαία και κίνηση. Παρόλα αυτά, υπάρχουν μεγάλες προκλήσεις στο verification και validation (V&V) της αξιολόγησης ασφάλειας. Λόγω της πιθανής μη προβλέψιμης φύσης της υπολογιστικής νοημοσύνης, κατάλληλες V&V μέθοδοι πρέπει να εξασφαλίσουν την ασφαλή αυτονομία.

Ατυχήματα στον δρόμο προκαλούνται τόσο συχνά, που πλέον θεωρούνται κομμάτι της καθημερινότητας. Παρόλα αυτά, ο μεγάλος αριθμός δυστυχημάτων, πάνω από 1 εκατομμύριο ετησίως, δεν πρέπει να αγνοηθεί. Σύμφωνα με στατιστικές αναλύσεις, το 75% όλων των ατυχημάτων στους δρόμους προκαλείται από το ανθρώπινο σφάλμα. Αυτός ο αριθμός δεν δείχνει να μειώνεται όσο ο άνθρωπος έχει τον πλήρη έλεγχο από το όχημά του. Αυτό συμβαίνει, επειδή από τη φύση του, παίρνει αποφάσεις κατά 90% βάσει της οπτικής αντίληψής του. Επίσης, είναι αδύνατον να γνωρίζει όλους τους κινδύνους γύρω του. Επιπλέον, και η κίνηση συνήθως προκαλείται από τον άνθρωπο από λάθη απροσεξίας. Ως αποτέλεσμα, δεν έχει μόνο το χάσιμο χρόνου και πόρων, αλλά και τη μόλυνση της ατμόσφαιρας και την αύξηση του φαινομένου του θερμοκηπίου. Τα αυτόνομα αυτοκίνητα φαίνεται να είναι μια εφικτή λύση για αυτά τα ζητήματα. Η ορθή λειτουργία τους, όμως, και ως συνέπεια η αυξημένη ασφάλεια σε σχέση με την ανθρώπινη οδήγηση, πρέπει πρώτα να αποδειχθεί.

Συστήματα υπολογιστικής νοημοσύνης έχουν φέρει εντυπωσιακά αποτελέσματα σε διάφορους τομείς και βελτιώνονται γρήγορα. Παρόλα αυτά, η χρήση τους σε τομείς ασφαλείας δημιουργούν αρκετά προβλήματα στο πλαίσιο του V&V, διότι δεν είναι πλήρως προβλέψιμα. Τέτοια συστήματα χαρακτηρίζονται μη ασφαλή μέχρι να αποδειχθεί το αντίθετο. Το ISO 26262, είναι μια παγκόσμια προδιαγραφή για την ασφάλεια σε ηλεκτρικών και ηλεκτρονικών συσκευών που ενσωματώνονται σε μεταφορικά μέσα. Αν πληρούνται οι προδιαγραφές του ISO 26262, τότε τα αυτόνομα αυτοκίνητα θεωρητικά μπορούν να χαρακτηριστούν ασφαλή. Παρόλα αυτά, δεν είναι ακόμα ξεκάθαρο αν αυτή η προδιαγραφή μπορεί να χρησιμοποιηθεί αποτελεσματικά για την αξιολόγηση τέτοιων συστημάτων.

A. Επίπεδα Αυτοματισμού

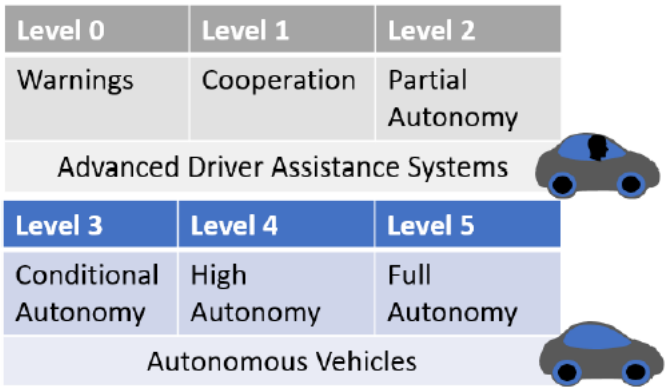


Figure 12. Επίπεδα Αυτοματισμού

Επιπλέον, έστω ότι ένα όχημα αποδώσει καλά σε διάφορες δοκιμές και με ικανοποιητικό ρυθμό σφαλμάτων, η χρήση εκατομμυρίων τέτοιων οχημάτων μπορεί να αυξήσει τον ρυθμό αυτόν σε μη επιτρεπτά επίπεδα. Μια αλλαγή σε μια τόσο μεγάλη κλίμακα είναι απίθανη. Μια πιο πιθανή προσέγγιση είναι η μείωση του ελέγχου του οδηγού σταδιακά έτσι, ώστε να επιτευχθεί τελικά ο πλήρης αυτοματισμός. Ο Σύλλογος Μηχανικών

Αυτοκινήτων (Society of Automobile Engineers - SAE) έχει ορίσει έξι επίπεδα αυτοματισμού για τα αυτόνομα αυτοκίνητα. Αυτά τα επίπεδα καθορίζουν τις τεχνικές απαιτήσεις για τη εξασφάλιση της ασφάλειάς τους.[3]

- Στο επίπεδο 0, ο οδηγός έχει τον πλήρη έλεγχο του οχήματος. Το όχημα μπορεί μόνο να ειδοποιήσει τον οδηγό σε περίπτωση κινδύνου και δεν μπορεί να ελέγξει τους ενεργοποιητές (actuators).
- Στο επίπεδο 1, ο οδηγός και το όχημα συνεργάζονται κατά την οδήγηση. Η συνεισφορά του οχήματος βελτιώνει την απόδοση της. Είναι, όμως, περιορισμένο στη χρήση του τιμονιού και στην επιτάχυνση. Ο οδηγός πρέπει να παραμένει πάντα σε ετοιμότητα.
- Στο επίπεδο 2, το όχημα βοηθάει τον οδηγό, έχοντας τον πλήρη έλεγχο. Αν και ο οδηγός δεν χρειάζεται να ελέγξει το όχημα, θα πρέπει να έχει επίγνωση της κατάστασης γύρω του, ώστε να μπορεί να παρέμβει όποτε χρειαστεί.
- Στο επίπεδο 3, τα οχήματα έχουν περιορισμένη αντίληψη και παίρνουν αποφάσεις μέσω αισθητήρων. Παρόλο που ο οδηγός δεν χρειάζεται να έχει πλήρη επίγνωση της κατάστασης γύρω του, πρέπει να μπορεί να πάρει τον έλεγχο σε επικίνδυνες καταστάσεις. Η προσοχή του οδηγού είναι χαμηλότερων απαιτήσεων σε αυτό το επίπεδο.
- Στο επίπεδο 4, παρόλο που ο οδηγός μπορεί να παρέμβει, δεν είναι αναμενόμενο να πάρει τον έλεγχο του οχήματος. Οχήματα υψηλού αυτοματισμού είναι υπεύθυνα για όλες τις διεργασίες και μπορούν να λειτουργήσουν με ασφάλεια σε προκαθορισμένες καταστάσεις (modes) χωρίς την βοήθεια του οδηγού.
- Στο επίπεδο 5, τα οχήματα λειτουργούν αυτόνομα σε οποιαδήποτε κατάσταση. Ο οδηγός είναι τελείως έξω από το κύκλωμα. Υπάρχει η δυνατότητα αφαίρεσης του τιμονιού και των πετάλιων.

Τα επίπεδα από 0 έως 2 χαρακτηρίζονται συνήθως ως Advanced Driver Assistance Systems (ADAS). Ο οδηγός είναι υπεύθυνος για τη λειτουργία του οχήματος και πρέπει να το επιβλέπει και να παρεμβαίνει όποτε χρειάζεται.

Μέχρι στιγμής, τα πιο προχωρημένα αυτόνομα αυτοκίνητα που έχουν παρουσιαστεί στο κοινό είναι μόνο μερικώς αυτόνομα, εφόσον ο οδηγός πρέπει να μένει σε εγρήγορση και να παρεμβαίνει σε επικίνδυνες καταστάσεις. Ένα μοιραίο δυστύχημα το 2018 μεταξύ ενός Τέσλα και μιας νταλίκας έδειξε πόσο σημαντική είναι η ανθρώπινη επίβλεψη στα αυτόνομα αυτοκίνητα στα τωρινά επίπεδα τεχνολογίας. Το ατύχημα συνέβη, διότι το σύστημα της υπολογιστικής όρασης του αυτοκινήτου δεν κατάφερε να εντοπίσει μια άσπρη νταλίκα σε ένα συννεφιασμένο τοπίο. Μια τέτοια συνθήκη δεν ήταν προϋπολογισμένη και τεσταρισμένη από την Τέσλα, κάτι που μόνο ένας άνθρωπος μπορούσε να εντοπίσει.

Figure 13. Δυστύχημα μεταξύ ενός Τέσλα και μιας νταλίκας

Η αυξημένη πολυπλοκότητα του λογισμικού αυξάνει το κόστος παραγωγής και αξιόπιστα V&V είναι απαραίτητα, γιατί έστω και ένα λάθος να περάσει απαρατήρητο μπορεί να προκαλέσει τεράστια ζημιά στην εταιρία. Ακόμα και χωρίς την εμπλοκή υπολογιστικής νοημοσύνης, το 60% των αποσύρσεων των οχημάτων είναι εξ αιτίας σφαλμάτων στο λογισμικό.

B. V-Model

Το ISO 26262 είναι ένα πρότυπο που ακολουθεί το λεγόμενο V-model για τη διαχείριση V&V , και παρέχει κατευθυντήριες γραμμές για τη σχεδίαση και ενσωμάτωση λογισμικού και εξαρτημάτων. Το V-model χρησιμοποιείται για την ανάπτυξη λογισμικού και testing για περισσότερα από 20 χρόνια.

Στο μοντέλο, η δεξιά μεριά περιγράφει την σταδιακά αυξανόμενη διαδικασία του V&V που εφαρμόζεται μετά το μοντέλο ανάπτυξης του καταρράκτη που φαίνεται στην αριστερή μεριά. Στα αυτόνομα αυτοκίνητα, οι διαδικασίες ανάπτυξης αντιστοιχίζονται με δυσκολία στο V-model, λόγω της μη προβλέψιμης φύσης της υπολογιστικής νοημοσύνης.

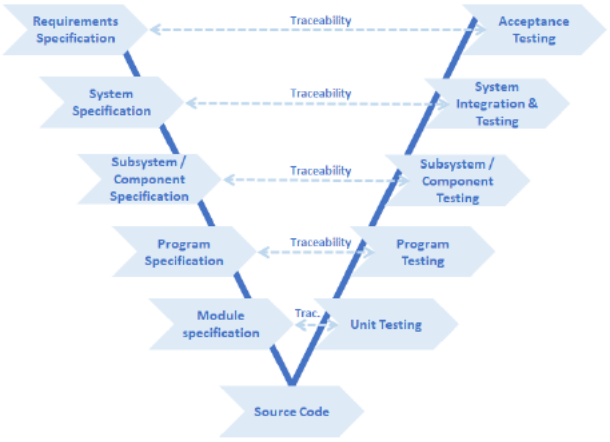


Figure 14. V-model in ISO 26262

V. Μεθοδολογίες VERIFICATION και VALIDATION σε Αυτόνομα Αυτοκίνητα

Η εξασφάλιση της ασφάλειας των αυτόνομων αυτοκινήτων θεωρείται διεπιστημονική πρόκληση. Είναι απαραίτητη η συνεργασία διάφορων ειδικοτήτων για μια αξιόπιστη σχεδίαση.

A. Μεθοδολογία *Monitor-Actuator*

Μια στρατηγική για την αξιολόγηση του ρίσκου, σύμφωνα με το πρότυπο ISO 26262, είναι η εφαρμογή μιας αρχιτεκτονικής monitor - actuator. Ο actuator εκτελεί τη βασική λειτουργία και η μονάδα παρακολούθησης (monitor) τη διαδικασία επαλήθευσης. Αν υπάρχει κάποιο πρόβλημα στον actuator, το σύστημα παρακολούθησης αναμένεται να σταματήσει ολόκληρη τη λειτουργία, με αποτέλεσμα η ύπαρξη ενός fail-safe συστήματος. Το σύστημα παρακολούθησης πρέπει να είναι ικανό να εντοπίζει τα σφάλματά του, ώστε να αποφευχθεί το σενάριο που ένα σφάλμα μέσα στο σύστημά του εμποδίζει την ανίχνευση σφάλματος στον actuator.

Η fail-safe λειτουργία του ζεύγους αυτού είναι ένα μεγάλο προνόμιο, όμως έχει μία αδυναμία. Σε περίπτωση κάποιου προβλήματος και της μη ορθής λειτουργίας του monitor, ο actuator παύει να λειτουργεί, πράγμα το οποίο δεν είναι πάντα επιθυμητό. Πχ. Ο τερματισμός λειτουργίας του τιμονιού, ενώ το όχημα ακόμα κινείται.

Η εφαρμογή αυτής της αρχιτεκτονικής πρέπει να γίνει με περισσότερα από ένα ζευγάρια monitor - actuator για cross - validation. Και η σχεδίαση θα πρέπει να διαφέρει από ζευγάρι σε ζευγάρι, ώστε σε περίπτωση σφάλματος του λογισμικού να μην καταρρεύσει ολόκληρο το σύστημα. Πχ. Ariane 5 Flight 501. Η ίδια εξαίρεση οδήγησε σε αποτυχία του κυρίου συστήματος αλλά και του εναλλακτικού, με συνέπεια να εκραγεί ο πύραυλος κατά την απογείωση. Αν και είναι σημαντικό να διαφέρουν τα εξαρτήματα μεταξύ τους, δεν είναι εύκολο να επιτευχθεί. Για τα ίδια προαπαιτούμενα είναι πολύ πιθανό και τα εξαρτήματα που σχεδιάζονται να έχουν τα ίδια σφάλματα. Εν κατακλείδι, μια αρχιτεκτονική που εγγυάται μια fail-safe λειτουργία εντοπίζοντας σφάλματα στο σύστημα είναι απαραίτητη για ασφαλή αυτόνομα συστήματα.

B. Μεθοδολογίες *Machine Learning* και *Deep Neural Networks*

Ένας άλλος τρόπος είναι με τη χρήση machine learning και deep neural networks. Test data του πραγματικού κόσμου κατηγοριοποιούνται βάσει της λειτουργικότητάς τους με τη χρήση ML και DNN αλγορίθμων. Με αυτά τα δεδομένα μπορούν να προσομοιωθούν κι άλλα δεδομένα και έτσι να εκπαιδευτεί το σύστημα που μελετάται. Επίσης όλα τα δεδομένα αποθηκεύονται σε βάσεις δεδομένων όπου και μπορούν να χρησιμοποιηθούν από διάφορους κατασκευαστές στις διάφορες φάσεις του V-model.

Τέλος, υπάρχει μια καινούρια μέθοδος σε ερευνητικό επίπεδο σύμφωνα με την οποία το αυτόνομο όχημα πρέπει να παίρνει μόνο αποφάσεις που είναι επαληθευμένες και σίγουρα ασφαλείς. Οπότε μπορεί να λειτουργεί μόνο σε ένα προκαθορισμένο safe space.Έτσι μπορεί αρχικά να βγει στον δρόμο ένα όχημα στο επίπεδο 2 και να το επιβλέπει ο οδηγός όποτε χρειάζεται και με τη πάροδο του χρόνου το όχημα να μαθαίνει και σταδιακά μεγαλώνει το safe space του ώστε τελικά να φτάσει σε επίπεδο 5 αυτοματισμού.

VI. VERIFICATION Αρχιτεκτονικής Λογισμικού

Η αρχιτεκτονική λογισμικού επηρεάζει σημαντικά την ανάπτυξη software-intensive συστημάτων [4]. Μια καλή αρχιτεκτονική καλύπτει τις ποιοτικές απαιτήσεις και επιτυγχάνει τους στόχους της σχεδίασης. Για να γίνουν πιο κατανοητές εκφράζονται με τη βοήθεια περιγραφών σε κάποια architecture description language (ADL). Αυτές οι περιγραφές μπορούν να χρησιμοποιηθούν ως μοντέλα κατά τη σχεδίαση ή την εκτέλεση του λογισμικού, αλλά και ως support documentation, για

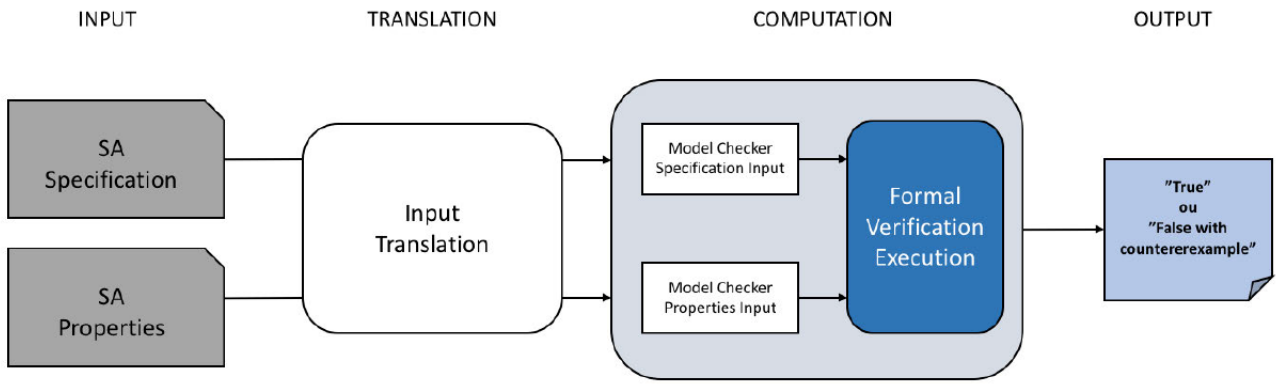


Figure 15. A process to formally verifying software architecture properties.

τη συντήρηση, την αξιολόγηση και την εξέλιξη του λογισμικού.

Οι ADL γλώσσες κατηγοριοποιούνται ως εξής:

- formal, με ακριβή σύνταξη και σημασιολογία, συνήθως μαθηματική, που υποστηρίζει αυτοματοποιημένη αρχιτεκτονική ανάλυση.
- semi-formal, με καλά ορισμένη σύνταξη, αλλά με ελλιπή σημασιολογία και
- informal, ad-hoc διαγράμματα, τα οποία δεν μπορούν να αναλυθούν επίσημα και περιορίζουν την χρησιμότητα της περιγραφής της αρχιτεκτονικής.

Μια από τις κύριες προκλήσεις στη σχεδίαση συστημάτων με βαρύ λογισμικό σχετίζεται με την ανάλυση της αρχιτεκτονικής, δηλαδή να εντοπιστούν σημαντικές ιδιότητες του λογισμικού πριν ακόμα ενσωματωθεί στο σύστημα χρησιμοποιώντας αρχιτεκτονικά μοντέλα. Η διαδικασία αυτή πρέπει να πραγματοποιείται όσο το δυνατόν νωρίτερα για την αποφυγή πιθανών σφαλμάτων και μη επιθυμητά αποτελέσματα. Σε προχωρημένες φάσεις της σχεδίασης η διόρθωση σφαλμάτων είναι ακριβή.

Η διαδικασία είναι απαραίτητη ειδικά σε περίπλοκα συστήματα, όπου η αρχιτεκτονική τους πρέπει να επαληθεύεται, ώστε να λειτουργούν σωστά χωρίς σφάλματα. Για αυτόν τον λόγο, formal περιγραφές αρχιτεκτονικής είναι επιθυμητές για την καλύτερη υποστήριξη αυτοματοποιημένης αρχιτεκτονικής ανάλυσης.

Το κύριο πλεονέκτημα του formal approach είναι ότι αποφασίζει αν το σύστημα λογισμικού μπορεί να ικανοποιήσει ιδιότητες και περιορισμούς σχετικά με τις προαπαιτήσεις και ελέγχει την ευστοχία και εγκυρότητα στη σχεδίαση αρχιτεκτονικής.

Μία από τις κυριότερες μεθόδους που χρησιμοποιούνται για το formal verification αρχιτεκτονικής λογισμικού είναι το model checking. Είναι μια εξαντλητική μέθοδος αυτόματης επαλήθευσης με βασικό στόχο να ελέγχει αν μια προδιαγραφή της αρχιτεκτονικής ικανοποιεί τις ιδιότητές της.

Δέχεται ως είσοδο τις προδιαγραφές της αρχιτεκτονικής του λογισμικού (πχ. μια περιγραφή σε ADL) και τις ιδιότητές της. Αν αυτές οι προδιαγραφές δεν είναι επίσημα γραμμένες, τότε απαιτείται πρώτα μια μετάφραση σε μια επίσημη μορφή. Έπειτα, περνούν οι ιδιότητες από την διαδικασία επαλήθευσης. Αν η έξοδος είναι true, τότε οι ιδιότητες ικανοποιούνται. Αν η έξοδος είναι false, τότε κάποια ιδιότητα παραβιάζεται.

Οι Ιδιότητες της αρχιτεκτονικής χωρίζονται σε τρεις βασικές κατηγορίες

- reliability concerns, ανησυχίες στην αξιοπιστία
- σημαντικές ιδιότητες σε παράλληλα συστήματα, όπως νομιμότητα, ασφάλεια, κρισιμότητα, προτεραιότητα και
- ιδιότητες συγκεκριμένες για συστήματα πραγματικού χρόνου

Οι ιδιότητες αυτών των τριών κατηγοριών βρίσκονται συνήθως σε critical συστήματα και δεν είναι τυχαία η έμφαση που τους δίνεται. Τα συστήματα αυτά είναι ηλεκτρονικά αεροσκάφους, ιατρικά συστήματα και σιδηροδρομικά.

References

[1] J. You, J. Li, and S. Xia, "A survey on formal methods using in software development," 2012.

[2] Y. Guan, J. Guo, and Q. Li, "Formal verification of a hybrid iot operating system model," *IEEE Access*, vol. 9, pp. 59 171–59 183, 2021.

[3] N. Rajabli, F. Flammini, R. Nardone, and V. Vittorini, “Software verification and validation of safe autonomous cars: A systematic literature review,” *IEEE Access*, 2020.
[4] C. Araujo, E. Cavalcante, T. Batista, M. Oliveira, and F. Oquendo, “A research landscape on formal verification of software architecture descriptions,” *IEEE Access*, vol. 7, pp. 171 752–171 764, 2019.