A Comprehensive Study of Declarative Modelling Languages

B, Event-B, Alloy, Dash, TLA$^+$, PlusCal, AsmetaL

Amin Bandali

June 30, 2020

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS
DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

A Comprehensive Study of
Declarative Modelling Languages
B, Event-B, Alloy, Dash, TLA+, PlusCal, AsmetaL
Amin Bandali
June 30, 2020
UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

- hello, welcome!
- i'm Amin Bandali, and today i'm presenting my master's thesis, A Comprehensive Study of Declarative Modelling Languages
- thank you to each and every one of you for being here
- especially Prof. Atlee and Prof. Rayside for agreeing to be my second readers and reading my thesis in such a small amount of time

Formal Specifications

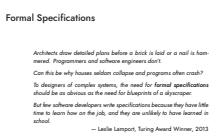
Architects draw detailed plans before a brick is laid or a nail is hammered. Programmers and software engineers don't.

Can this be why houses seldom collapse and programs often crash?

To designers of complex systems, the need for ***formal specifications*** *should be as obvious as the need for blueprints of a skyscraper.*

But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.

— Leslie Lamport, Turing Award Winner, 2013


Formal Specifications

Architects draw detailed plans before a brick is laid or a nail is hammered. Programmers and software engineers don't.

Can this be why houses seldom collapse and programs often crash?

To designers of complex systems, the need for **formal specifications** should be as obvious as the need for blueprints of a skyscraper.

But few software developers write specifications because they have little time to learn how on the job, and they are unlikely to have learned in school.

— Leslie Lamport, Turing Award Winner, 2013

1. i'd like to start my presentation with a quote from Leslie Lamport about formal specifications, a shorter excerpt of which i used in my first chapter's epigraph
2. *read the quote...*
3. with this quote, Lamport makes the point for learning and using formal specifications as an important tool for software developers and especially software engineers

Declarative Behavioural Modelling

Declarative behavioural modelling is a powerful modelling paradigm that enables users to model system functionality abstractly and formally.

An *abstract model* is a concise and compact representation of the key characteristics of a system, and enables the stakeholders to reason about the correctness of the system in the early stages of development.

Declarative Behavioural Modelling

Declarative behavioural modelling is a powerful modelling paradigm that enables users to model system functionality abstractly and formally.

An *abstract model* is a concise and compact representation of the key characteristics of a system, and enables the stakeholders to reason about the correctness of the system in the early stages of development.

in this work we focus on the declarative behavioural modelling approach for formal specification

Declarative Modelling Languages

- are used to write behavioural formal specifications of systems
- using a state-machine-oriented / transition system approach
- in essence model a Kripke structure

Examples of declarative modelling languages:

- Alloy
- TLA$^+$
- VDM
- Z

Declarative Modelling Languages
- are used to write behavioural formal specifications of systems
- using a state-machine-oriented / transition system approach
- in essence model a Kripke structure

Examples of declarative modelling languages:
- Alloy
- TLA$^+$
- VDM
- Z

Declarative Models

- describe the transitions declaratively using constraints, rather than imperative calculations and/or statements;
- include user-defined and -axiomatized units of data, which can represent rich datatypes such as lists and trees;
- have a formal mathematical and logical foundation, usually first-order logic (FOL) and/or set theory; and
- allow writing models without specifying the size of sets (the scopes); the scopes may need to be specified for analysis.

Declarative Models
- describe the transitions declaratively using constraints, rather than imperative calculations and/or statements;
- include user-defined and -axiomatized units of data, which can represent rich datatypes such as lists and trees;
- have a formal mathematical and logical foundation, usually first-order logic (FOL) and/or set theory; and
- allow writing models without specifying the size of sets (the scopes); the scopes may need to be specified for analysis.

Use of Declarative Models

- Zave's use of Alloy and Spin to find specification-level bugs in the specification of the Chord network protocol;
- Amazon's use of TLA$^+$ has helped find subtle bugs in complex real-world systems and prevent the bugs from reaching production; and
- Huynh *et al.*'s use of B for formalizing a new healthcare access control model with conflict resolution for overriding patient consent as to who can access their Electronic Health Records (EHR) under strictly defined scenarios by regional laws of Québec and Canada to protect the patient's life.

Use of Declarative Models
- Zave's use of Alloy and Spin to find specification-level bugs in the specification of the Chord network protocol;
- Amazon's use of TLA$^+$ has helped find subtle bugs in complex real-world systems and prevent the bugs from reaching production; and
- Huynh *et al.*'s use of B for formalizing a new healthcare access control model with conflict resolution for overriding patient consent as to who can access their Electronic Health Records (EHR) under strictly defined scenarios by regional laws of Québec and Canada to protect the patient's life.

we are motivated to do this study by the many applications and demonstrated usefulness of declarative modelling languages and model checking to help design systems or analyze and verify properties about the design of existing systems

Research Question

Given that there are a great many declarative modelling languages to choose from, *how does one make a choice of which language to use*?

Research Question

Given that there are a great many declarative modelling languages to choose from, *how does one make a choice of which language to use*?

Thesis Contributions

The contributions of this thesis are

- a set of criteria to compare declarative modelling languages;
- comparison of selected declarative modelling languages (B, Event-B, Alloy, Dash, TLA$^+$, PlusCal, and AsmetaL) based on these criteria; and
- recommendations for the choice of modelling language based on the characteristics of the transition system under description.

Thesis Contributions

The contributions of this thesis are
- a set of criteria to compare declarative modelling languages;
- comparison of selected declarative modelling languages (B, Event-B, Alloy, Dash, TLA$^+$, PlusCal, and AsmetaL) based on these criteria; and
- recommendations for the choice of modelling language based on the characteristics of the transition system under description.

Methodology

1. select 6 relatively small examples across the data- vs. control-oriented spectrum;
   - *control-oriented*: has complex conditions for when a transition is relevant that are naturally expressed using modes, control states, or concurrency; and
   - *data-oriented*: has complex constructions of data;
2. model the examples in the 3 languages B, Dash, and TLA$^+$;
3. describe the differences and similarities across the languages while modelling the examples, forming the initial comparison criteria for comparing the languages; and
4. publish our results.

Methodology
1. select 6 relatively small examples across the data- vs. control-oriented spectrum;
   - control-oriented: has complex conditions for when a transition is relevant that are naturally expressed using modes, control states, or concurrency; and
   - data-oriented: has complex constructions of data;
2. model the examples in the 3 languages B, Dash, and TLA+;
3. describe the differences and similarities across the languages while modelling the examples, forming the initial comparison criteria for comparing the languages; and
4. publish our results.

Methodology (cont'd)

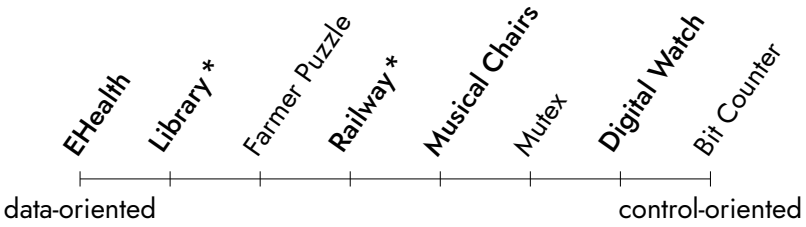
5. expand set of comparison criteria to include other interesting characteristics of languages;
6. expand our set of chosen languages to include AsmetaL, Alloy, Event-B, and PlusCal in addition to B, Dash, and TLA$^+$;
7. model existing examples in new languages, ensuring proper setup of tool support for all of the languages;

Methodology (cont'd)

5. expand set of comparison criteria to include other interesting characteristics of languages;
6. expand our set of chosen languages to include AsmetaL, Alloy, Event-B, and PlusCal in addition to B, Dash, and TLA$^+$;
7. model existing examples in new languages, ensuring proper setup of tool support for all of the languages;

Methodology (cont'd)

8. expand our set of examples with 2 new larger systems, Library and Railway (marked with an asterisk), increasing our model count from 18 relatively small examples to 35 including the larger ones; our final case studies are those typeset in **bold**; and



Methodology (cont'd)
8. expand our set of examples with 2 new larger systems, Library and Railway (marked with an asterisk), increasing our model count from 18 relatively small examples to 35 including the larger ones; our final case studies are those typeset in bold; and
EHealth Library * Farmer Puzzle Railway * Musical Chairs Mutex Digital Watch Bit Counter
data-oriented control-oriented

the figure shows the data- vs. control-oriented characterization spectrum

Methodology (cont'd)

Table 1: Order of modelling case studies across languages

Case study \ Language	B	Event-B	Alloy	Dash	TLA$^+$	PlusCal	AsmetaL
EHealth	1	2	3	1	1	4	5
Musical Chairs	1	4	E	1	1	3	2
Digital Watch	1	2	5	1	1	4	3
Library	E	1	E	3	5	2	4
Railway	1	7	3	5	6	4	2

Legend: E indicates Existing models, *i.e.* those that we had no influence on. The numbers in each row indicate the order of languages the case study was done in.

9. note the differences and similarities across the languages with respect to our comparison criteria while modelling the examples.

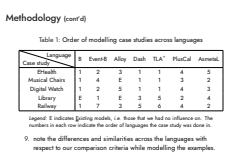

Methodology (cont'd)

Table 1: Order of modelling case studies across languages

Case study \ Language	B	Event-B	Alloy	Dash	TLA$^+$	PlusCal	AsmetaL
EHealth	1	2	3	1	1	4	5
Musical Chairs	1	4	E	1	1	3	2
Digital Watch	1	2	5	1	1	4	3
Library	E	1	E	3	5	2	4
Railway	1	7	3	5	6	4	2

Legend: E indicates Existing models, *i.e.* those that we had no influence on. The numbers in each row indicate the order of languages the case study was done in.

9. note the differences and similarities across the languages with respect to our comparison criteria while modelling the examples.

└Methodology (cont'd)

the table shows the order of modelling each of the case studies across the languages

Comparison Criteria

We classify our comparison criteria into 3 main categories:

- **control modelling** (structuring transition systems),
- **data modelling** (data descriptions in transition systems), and
- **modularity** aspects of modelling.

Comparison Criteria

We classify our comparison criteria into 3 main categories:
- control modelling (structuring transition systems),
- data modelling (data descriptions in transition systems), and
- modularity aspects of modelling.

Introduction & Motivation
Methodology
Comparison Criteria
Control Modelling
Data Modelling
Modularity
Contributions

Control Modelling

concerned with control aspects and structure of transition systems

Control Modelling

concerned with control aspects and structure of transition systems

Control Modelling

concerned with control aspects and structure of transition systems

a transition system TS is a tuple (S, TR, I), where

- S is a set of snapshots,
- $TR \subseteq S \times S$ is a transition relation, and
- $I \subseteq S$: is a set of initial snapshots.

Control Modelling

concerned with control aspects and structure of transition systems

a transition system TS is a tuple (S, TR, I), where

- S is a set of snapshots,
- $TR \subseteq S \times S$ is a transition relation, and
- $I \subseteq S$: is a set of initial snapshots.

A model in a declarative modelling language defines a transition system that starts in an initial snapshot $s_0 \in I$ and progresses from a snapshot s to the next snapshot s' for $(s, s') \in TR$.

Control Modelling Criteria

- snapshot variables
- initialization
- **transition relation**
- **control state hierarchy**
- invariants
- inconsistency
- **frame problem**

Control Modelling Criteria
- snapshot variables
- initialization
- transition relation
- control state hierarchy
- invariants
- inconsistency
- frame problem

- our criteria for control aspects of models are: ...
- in the interest of time we will focus on the **bold** ones in this presentation which we thought might be more interesting than the others
- if asked about inconsistency, elaborate:
  - deadlock
  - contradictory TR
  - contradictory TP
  - stuttering

Control Modelling — Transition Relation

- completely explicit: Alloy
- mostly explicit: TLA$^+$ and AsmetaL
- implicit: B, Event-B, Dash, and PlusCal

Control Modelling — Transition Relation

- completely explicit: Alloy
- mostly explicit: TLA$^+$ and AsmetaL
- implicit: B, Event-B, Dash, and PlusCal

- in Alloy, TR is defined *completely explicitly* in model text, and its form can vary greatly depending on how the snapshot, variables, and transitions are defined. *e.g.* with a `State` signature as the snapshot representation and its fields as variables, TR can be decomposed into `pred`icates which can be viewed as transitions.
- in TLA$^+$, TR is by convention a predicate named `Next`, defined as the disjunction of all of the model's transition predicates (method best supported by TLC, TLA$^+$'s accompanying MC). AsmetaL has a more imperative style, and does not have a disjunction operator for combining transitions; and as such, we have have to use the `choose` rule instead. TR said to be defined *mostly explicitly* because in addition to the model text, both languages add implicit stuttering under certain conditions.
- the remaining languages have *implicit* TR, constructed automatically behind the scenes from the transitions. it's worth mentioning that PlusCal allows writing one's own TR if need to.

TR in B, Event-B, and PlusCal is implicitly formed as follows: at any step, any transition whose precondition is satisfied (*i.e.* is enabled) may be chosen to be taken. There is no requirement on the preconditions of the transition to be non-overlapping, and more than one trans may be enabled at a time, resulting in a branch in the snapshot space graph.

Control Modelling — Control State Hierarchy

- **labelled control state**: is a distinguished set of variables with a finite set of values, that are used to control when a transition can be taken
- languages with labelled control states can have **control state hierarchy** and concurrency
- control state hierarchies are a powerful tool for representing the states or modes of a control-oriented system
- in our set of languages, unique feature of Dash

Control Modelling — Control State Hierarchy
- **labelled control state**: is a distinguished set of variables with a finite set of values, that are used to control when a transition can be taken
- languages with labelled control states can have **control state hierarchy** and concurrency
- control state hierarchies are a powerful tool for representing the states or modes of a control-oriented system
- in our set of languages, unique feature of Dash

Control Modelling — Frame Problem

refers to the issue of how snapshot variables that are not explicitly constrained in a transition may or may not change from one snapshot to the next

Control Modelling — Frame Problem

refers to the issue of how snapshot variables that are not explicitly constrained in a transition may or may not change from one snapshot to the next

is particularly an issue in declarative languages that rely on logical constraints on variables for describing the changed and unchanged variables in a transition

Control Modelling — Frame Problem

- Alloy: all unconstrained variables may change
- B, Event-B, and PlusCal: all unconstrained variables remain unchanged
- Dash and AsmetaL: monitored (environmental) variables may change from one snapshot to the next, controlled variables not constrained in a transition remain unchanged by it
- TLA$^+$: requires all transitions to constrain all variables, either by constraints on primed and unprimed names of variables or by marking them with the `UNCHANGED` keyword

Control Modelling — Frame Problem
- Alloy: all unconstrained variables may change
- B, Event-B, and PlusCal: all unconstrained variables remain unchanged
- Dash and AsmetaL: monitored (environmental) variables may change from one snapshot to the next, controlled variables not constrained in a transition remain unchanged by it
- TLA$^+$: requires all transitions to constrain all variables, either by constraints on primed and unprimed names of variables or by marking them with the UNCHANGED keyword

Data Modelling

concerned with the description of the data aspects of models

Data Modelling

concerned with the description of the data aspects of models

Data Modelling Criteria

- primitives & subtypes
- **constructors**
- built-ins
- expressions
- events
- constants
- well-formedness & **typechecking**
- scopes

Data Modelling Criteria
- primitives & subtypes
- constructors
- built-ins
- expressions
- events
- constants
- well-formedness & typechecking
- scopes

- our criteria for data aspects of models are: ...
- we will focus on the **bold** ones in this presentation for similar reasons to Control Modelling earlier
- primitives in all languages consist of scalars and sets, with the exception of Alloy, which does not have scalars and "scalars" are represented using singleton sets

Data Modelling — Constructors

are operators that create *composite units of data* from primitives or other composite data units

examples: functions, relations, and records

constructors may include *multiplicities*, which impose constraints limiting the values in the composite data being constructed

Data Modelling — Constructors

are operators that create composite units of data from primitives or other composite data units

examples: functions, relations, and records

constructors may include multiplicities, which impose constraints limiting the values in the composite data being constructed

Data Modelling — Constructors

examples:

- B and Event-B have *arrow* constructors for creating functions; *e.g.*
  - $\nrightarrow$ and $\rightarrow$ for partial and total functions
  - $\twoheadrightarrow$ and $\twoheadrightarrow$ for partial and total surjective functions
- Alloy and Dash have *multiplicity keywords* such as `lone`, `one`, and `some` that can be used to create various kinds of functions; *e.g.*
  - `->lone` and `->one` for partial and total functions
  - `some->lone` and `some->one` for partial and total surjective functions

Data Modelling — Constructors
examples:
• B and Event-B have arrow constructors for creating functions; e.g.
  • ⇸ and → for partial and total functions
  • ⤔ and ↠ for partial and total surjective functions
• Alloy and Dash have multiplicity keywords such as lone, one, and some that can be used to create various kinds of functions; e.g.
  • ->lone and ->one for partial and total functions
  • some->lone and some->one for partial and total surjective functions

the -> operator in Alloy (and Dash) is actually the relation constructor, and the multiplicity keywords can constrain the constructed relation *e.g.* to be a function

Data Modelling — Typechecking

is the process of checking whether the use of data in the formulas conforms to the constraints expressed in the *type signatures*

type signatures are syntax denoting the kind of object contained in a snapshot variable, constant, or quantified variable as either a primitive or a composite unit of data

Data Modelling — Typechecking

is the process of checking whether the use of data in the formulas conforms to the constraints expressed in the *type signatures*

type signatures are syntax denoting the kind of object contained in a snapshot variable, constant, or quantified variable as either a primitive or a composite unit of data

Data Modelling — Typechecking

- TLA$^+$ and PlusCal have no type signatures, and typing constraints are stated and checked along with other invariants
- in Alloy and Dash typechecking consists of checking that no relation has been given different arities and that no expression can be shown to be redundant or contain a redundant sub-expression using solely the declarations
- B, Event-B, and AsmetaL have type signatures and typechecking that help statically catch errors like assigning a value from a set to a variable with a type signature declaring a different/incompatible set, and applying a function to arguments that do not match its type signature

Data Modelling — Typechecking
- TLA$^+$ and PlusCal have no type signatures, and typing constraints are stated and checked along with other invariants
- in Alloy and Dash typechecking consists of checking that no relation has been given different arities and that no expression can be shown to be redundant or contain a redundant sub-expression using solely the declarations
- B, Event-B, and AsmetaL have type signatures and typechecking that help statically catch errors like assigning a value from a set to a variable with a type signature declaring a different/incompatible set, and applying a function to arguments that do not match its type signature

a common example of the second form of type error in Alloy/Dash is an expression being redundant due to being equal to the empty relation (*e.g.* due to mismatched type signatures)

Modularity

concerned with the constructs of each language for writing modular descriptions of transition systems

Modularity

concerned with the constructs of each language for writing modular descriptions of transition systems

Modularity Criteria

- decomposition into **subtransition relations**
- namespaces of subtransition relations
- data decomposition into multiple files
- file import
- file export
- file parameterization
- file namespaces
- syntax overloading

Modularity Criteria
- decomposition into subtransition relations
- namespaces of subtransition relations
- data decomposition into multiple files
- file import
- file export
- file parameterization
- file namespaces
- syntax overloading

- our criteria for modularity aspects of models are: ...
- all of the languages allow data decomposition; *i.e.* allow subformulas relevant to the data aspects of the model, such as axioms for a unit of data, to be declared separately
- but we will focus on decomposition into subtransition relations in this presentation

Modularity — Subtransition Relations

- a subtransition system is a full description of a transition system
- subtransition systems are composed to create the single top-level transition relation implicitly or explicitly
- B, Alloy, TLA$^+$, and AsmetaL support decomposition into subtransition systems

Modularity — Subtransition Relations

- a subtransition system is a full description of a transition system
- subtransition systems are composed to create the single top-level transition relation implicitly or explicitly
- B, Alloy, TLA$^+$, and AsmetaL support decomposition into subtransition systems

Alloy, TLA$^+$, and AsmetaL have explicit representation of TR, while B has implicit representation of TR and achieves subtransition system decomposition by effectively prepending the components of the subtransition system(s) to those of the parent transition system to compose the resulting final transition system

for contributions, in addition to the set of comparison criteria and comparing the languages with respect to those criteria, we offer recommendations for the choice of modelling language, the research question we set out to answer

Recommendations

- Alloy, TLA$^+$, and AsmetaL allow fine control over the **transition relation** of the transition system, due to their explicit representation for the transition relation
- Dash is a great choice for modelling control-oriented systems, such as the digital watch case study, where relevance of **transitions** and their being enabled can be captured using potentially hierarchical **control states**, **events**, and concurrent regions
- Alloy models may suffer from **inconsistency** due to under-specification of behaviours, relating to the **frame problem**, as we saw in the EHealth and Library case studies; TLA$^+$ addresses this using the `UNCHANGED` keyword

Recommendations
- Alloy, TLA$^+$, and AsmetaL allow fine control over the **transition relation** of the transition system, due to their explicit representation for the transition relation
- Dash is a great choice for modelling control-oriented systems, such as the digital watch case study, where relevance of **transitions** and their being enabled can be captured using potentially hierarchical **control states**, **events**, and concurrent regions
- Alloy models may suffer from **inconsistency** due to under-specification of behaviours, relating to the **frame problem**, as we saw in the EHealth and Library case studies; TLA$^+$ addresses this using the `UNCHANGED` keyword

Recommendations (cont'd)

- Dash and AsmetaL allow a clear distinction between the system and its surrounding environment by distinguishing between **controlled** and **monitored** variables
- B and Event-B's extensive set of arrow **constructors** for various relational and functional units of data allow terse and concise descriptions of highly-constrained composite units of data in highly data-oriented systems, such as the Library case study, both in **type signatures** and in formulas, similar to Alloy and Dash's **multiplicities**

Recommendations (cont'd)
- Dash and AsmetaL allow a clear distinction between the system and its surrounding environment by distinguishing between **controlled** and **monitored** variables
- B and Event-B's extensive set of arrow **constructors** for various relational and functional units of data allow terse and concise descriptions of highly-constrained composite units of data in highly data-oriented systems, such as the Library case study, both in **type signatures** and in formulas, similar to Alloy and Dash's **multiplicities**

Recommendations (cont'd)

- B and Event-B are similar languages with the same roots, but they differ in B's support for several kinds of relationships between machines, B's sequence and tree **built-ins** and record **constructor**, Event-B's additional arrows for constructing specific kinds of relations, and Event-B's support for declaring **subtypes** and partitioning a set into multiple disjoint subsets;
- PlusCal enjoys the power and expressiveness of TLA$^+$'s **expressions**, wrapped in a syntax geared more towards modelling multi-process concurrent and parallel algorithms, with additional **well-formedness** safeguards making certain classes of bugs unrepresentable in a valid model

Recommendations (cont'd)
- B and Event-B are similar languages with the same roots, but they differ in B's support for several kinds of relationships between machines, B's sequence and tree built-ins and record constructor, Event-B's additional arrows for constructing specific kinds of relations, and Event-B's support for declaring subtypes and partitioning a set into multiple disjoint subsets;
- PlusCal enjoys the power and expressiveness of TLA+'s expressions, wrapped in a syntax geared more towards modelling multi-process concurrent and parallel algorithms, with additional well-formedness safeguards making certain classes of bugs unrepresentable in a valid model

Recommendations (cont'd)

- with respect to modularity, Event-B, Dash, and PlusCal allow **data decomposition** of a model across multiple files, while B, Alloy, TLA$^+$, and AsmetaL additionally allow **control decomposition** of a model into subtransition systems across multiple files.

Recommendations (cont'd)
- with respect to modularity, Event-B, Dash, and PlusCal allow **data decomposition** of a model across multiple files, while B, Alloy, TLA$^+$, and AsmetaL additionally allow **control decomposition** of a model into subtransition systems across multiple files.

Thesis Contributions

The contributions of this thesis are

- a set of criteria to compare declarative modelling languages;
- comparison of selected declarative modelling languages (B, Event-B, Alloy, Dash, TLA$^+$, PlusCal, and AsmetaL) based on these criteria; and
- recommendations for the choice of modelling language based on the characteristics of the transition system under description.

Thesis Contributions

The contributions of this thesis are
- a set of criteria to compare declarative modelling languages;
- comparison of selected declarative modelling languages (B, Event-B, Alloy, Dash, TLA$^+$, PlusCal, and AsmetaL) based on these criteria; and
- recommendations for the choice of modelling language based on the characteristics of the transition system under description.

Thanks!

Thanks!

Lines of Code for Case Studies

Table 2: Lines of code for each case study across the languages

Case study \ Language	B	Event-B	Alloy	Dash	TLA$^+$	PlusCal	AsmetaL
EHealth	62	111	135	95	120	101	87
Digital Watch	135	210	295	112	197	160	142
Musical Chairs	68	84	130	65	101	106	97
Library Management	180	164	317	120	146	151	207
Railway	82	86	387	280	79	84	78

Lines of Code for Case Studies

Table 2: Lines of code for each case study across the languages

Case study \ Language	B	Event-B	Alloy	Dash	TLA$^+$	PlusCal	AsmetaL
EHealth	62	111	135	95	120	101	87
Digital Watch	135	210	295	112	197	160	142
Musical Chairs	68	84	130	65	101	106	97
Library Management	180	164	317	120	146	151	207
Railway	82	86	387	280	79	84	78