Exploring Formal Verification for First-Order Linear Temporal Logic Modulo Theories

José João Alves dos Santos Ferreira - 99259

Instituto Superior Técnico
Universidade de Lisboa

PIC2 - Master in Computer Science and Engineering
Advisor: José Fragoso Santos, Alessandro Gianola

Linear Temporal Logic

LTL is a modal temporal logic

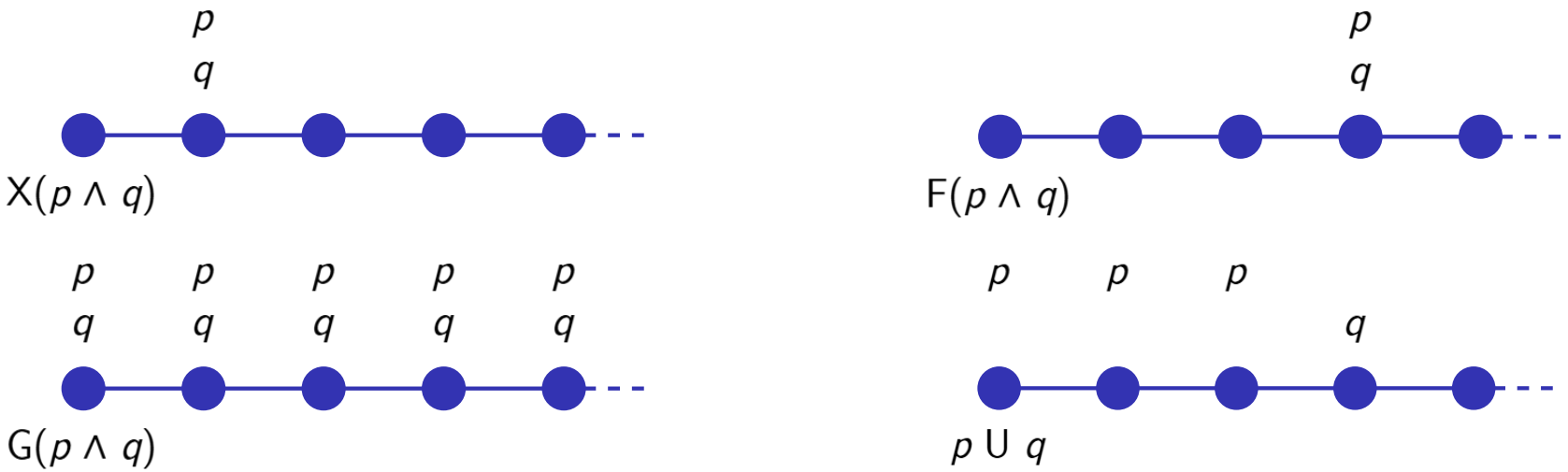
- interpreted over *discrete* state sequences
- it extends classical propositional logic
- temporal operators are used to talk about how propositions change over time

Linear Temporal Logic

LTL is a modal temporal logic

- interpreted over *discrete* state sequences
- it extends classical propositional logic
- temporal operators are used to talk about how propositions change over time

Semantics:



Linear Temporal Logic

$$\mathsf{G}(p) \wedge \neg q \wedge \mathsf{X}(\neg q) \wedge \mathsf{F}(q)$$

LTL

software model checking

chip design & hardware verification

network protocol verification

smart contracts in blockchain

First-Order Linear Temporal Logic

FOLTL is an extension of LTL

- it incorporates first-order logic, allowing the use of theories, predicates (e.g., $p(t_1, t_2)$), functions (e.g., $f(t_1, t_2, t_3)$) and variables (e.g., x, y) within temporal expressions
- it provides a richer framework for formal verification, capturing more complex dependencies than propositional logic

First-Order Linear Temporal Logic

FOLTL is an extension of LTL

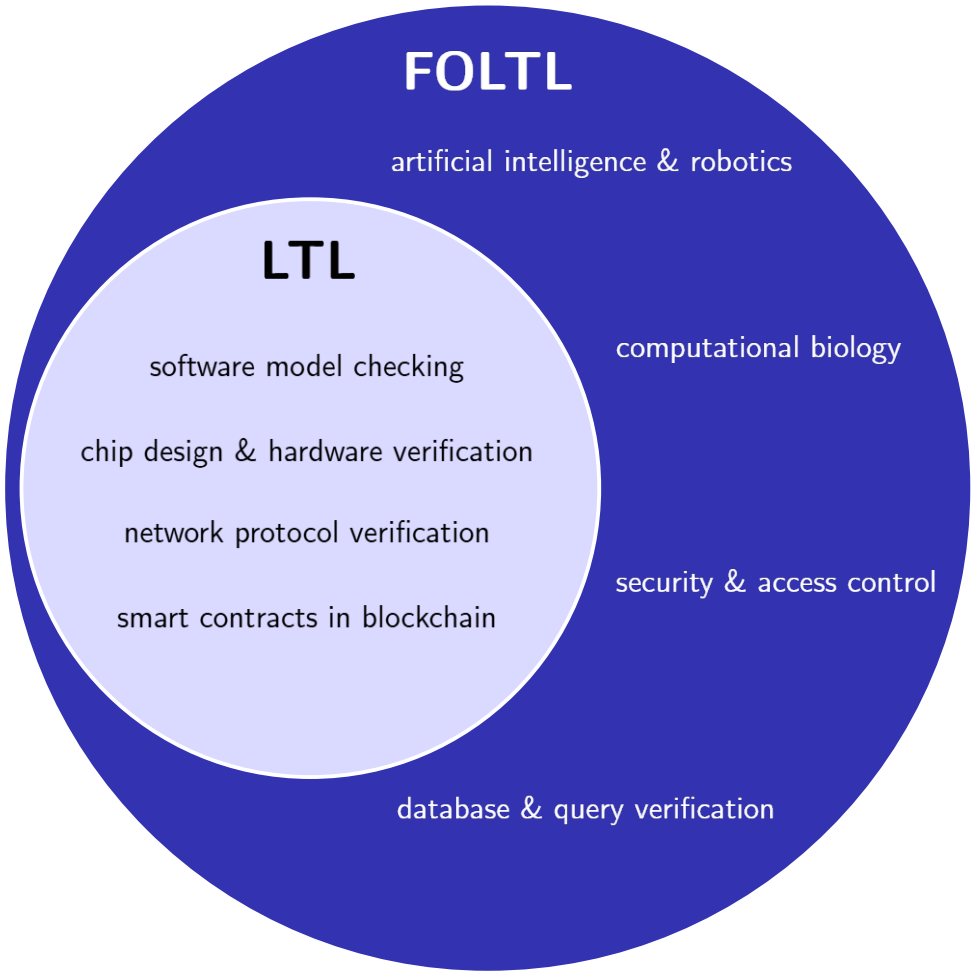
- it incorporates first-order logic, allowing the use of theories, predicates (e.g., $p(t_1, t_2)$), functions (e.g., $f(t_1, t_2, t_3)$) and variables (e.g., x, y) within temporal expressions
- it provides a richer framework for formal verification, capturing more complex dependencies than propositional logic

Theories: extensions added on top of first-order logics
(e.g., LIA, LRA, EUF, arrays, bit-vectors, strings)

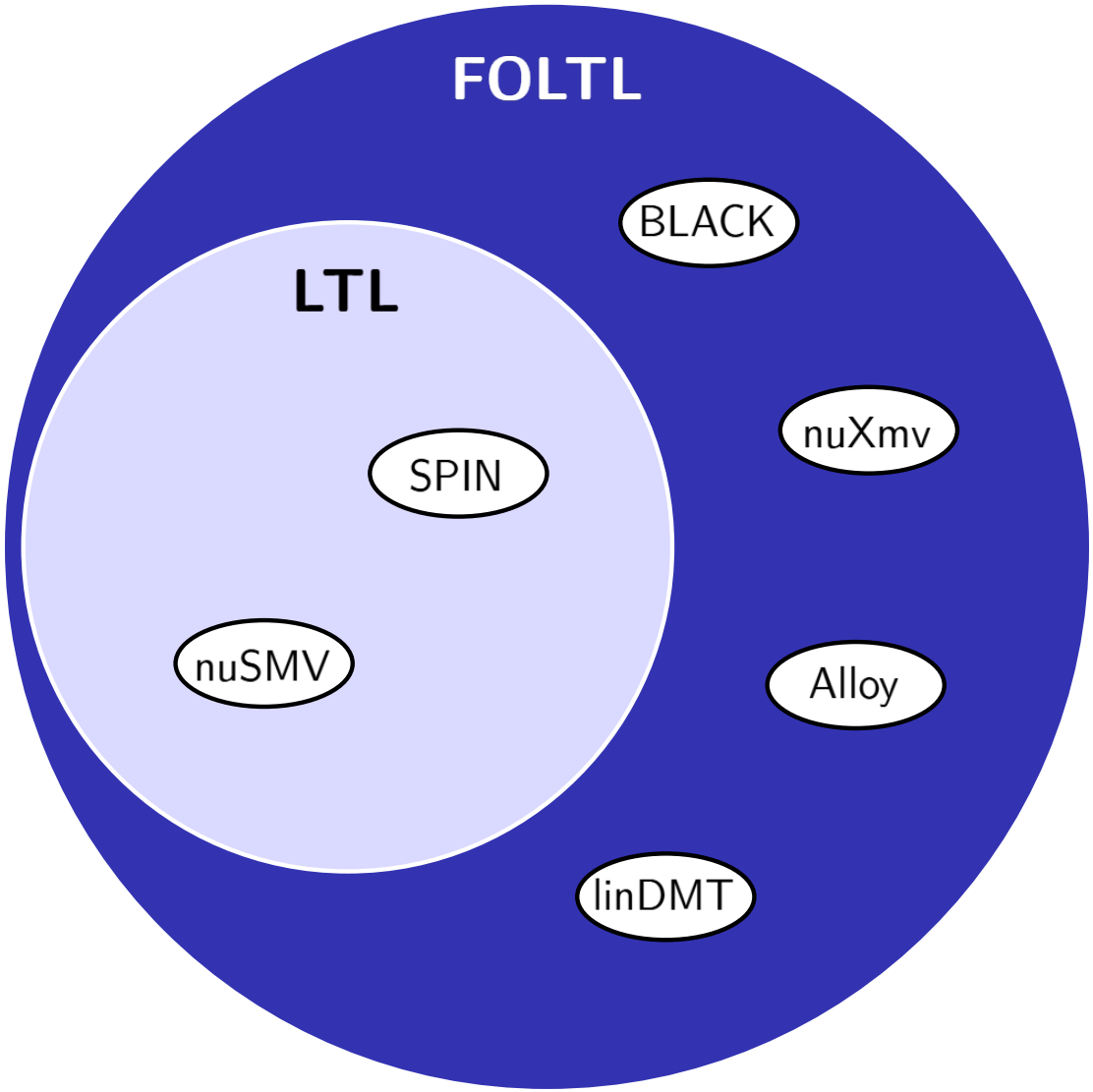
$$x = 0 \wedge \mathsf{F}(x = 1)$$

First-Order Linear Temporal Logic

$$\mathsf{G}(\mathit{IsEmail}(e) \wedge \mathit{IsPerson}(p)) \wedge (\mathit{BeingWritten}(e,p)\ \mathsf{U}\ (\mathit{Sent}(e,p) \wedge \mathsf{X}(\mathit{Delivered}(e) \vee \mathit{Failed}(e))))$$



Tools for FOLTL



Challenges

- Different tools support different FOLTL fragments and are designed for specific use cases
- They vary in how they perform formal verification
- There is no comparative study on the strengths and weaknesses of these tools

Challenges

- Different tools support different FOLTL fragments and are designed for specific use cases
- They vary in how they perform formal verification
- There is no comparative study on the strengths and weaknesses of these tools

What tool is the most suitable for a given specific task?

Contribution

- A comprehensive benchmark of formulas for assessing the most relevant FOLTL verification tools
- An empirical study to compare the expressiveness, effectiveness, and performance of these tools

Tools for FOLTL verification

	Typology	Time horizon	First-order domains	Primed variables	Quantifiers
BLACK					
nuXmv					
Alloy Analyzer					
linDMT					

Time horizon

Time is considered *linear*, *qualitative*, and *discrete*, interpreted over the natural numbers ($\mathbb{N}$).

Tools for FOLTL verification

Time horizon

Time is considered *linear*, *qualitative*, and *discrete*, interpreted over the natural numbers ($\mathbb{N}$).

FOLTL tools differ in their treatment of the time horizon: some assume *infinite* time, others handle *finite* time, and some support both.

Infinite words

$\bar{\sigma} = \langle \sigma_0, \sigma_1, \cdots \rangle$

Finite words

$\bar{\sigma} = \langle \sigma_0, \cdots, \sigma_n \rangle$

Tools for FOLTL verification

First-order domains

Domains emerge as theories are added to the logic. They can be *unbounded*, allowing variables to take unlimited values, or *bounded*, restricting them to a finite set.

Tools for FOLTL verification

First-order domains

Domains emerge as theories are added to the logic. They can be *unbounded*, allowing variables to take unlimited values, or *bounded*, restricting them to a finite set.

Unbounded domains

$$\mathbb{N} = \{0, 1, 2, \dots\}$$
$$\mathbb{R} = (-\infty, \infty)$$

enhance expressiveness but risk undecidability

Bounded domains

$$\text{State} = \{on, off, standby\}$$

simplify reasoning and ensure decidability

Primed variables

Given a free first-order variable x, its primed version x' represents the value of x in the next instant.

Primed variables

Given a free first-order variable x, its primed version x' represents the value of x in the next instant.

Ability to express the next value of x in terms of the current one:

Without the prime

$$\mathsf{X}(x = x)$$

With the prime

$$x' = x + 1$$

Quantifiers

FOLTL tools may or may not incorporate quantifiers over variable symbols. This enables reasoning about properties that hold for some or all elements in a domain.

Existential quantifier

$$\mathsf{G}(\exists \mathit{person} \,.\, \mathit{IsTheTallest}(\mathit{person}))$$

Universal quantifier

$$\forall \mathit{person} \,.\, \mathit{IsEating}(\mathit{person}) \rightarrow \mathsf{X}(\neg \mathit{IsHungry}(\mathit{person}))$$

Tools for FOLTL verification

BLACK

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
satisfiability checker	finite/ infinite	unbounded	✓	✓

Tools for FOLTL verification

BLACK

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
satisfiability checker	finite/ infinite	unbounded	✓	✓

- Encodes tableau branches of ϕ into SAT/SMT formulas, for increasing depths k, until an accepted branch or unsatisfiability witness is found
- Used for formal verification and artificial intelligence

Tools for FOLTL verification

 **nuXmv**

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
model checker	finite/ infinite	bounded/ unbounded	✓	×

Tools for FOLTL verification

 **nuXmv**

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
model checker	finite/ infinite	bounded/ unbounded	✓	×

- Uses tableau methods synchronously composed with the model
- Used in safety assessments, hybrid and railway interlocking systems

Tools for FOLTL verification

Alloy Analyzer

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
model checker	finite/ infinite	bounded	✓	×

Tools for FOLTL verification

Alloy Analyzer

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
model checker	finite/ infinite	bounded	✓	×

- Uses relational logic, an extension of first-order logic that excels at expressing structural properties
- Used for software and network design exploration

Tools for FOLTL verification

linDMT

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
model checker	finite	unbounded	✓	×

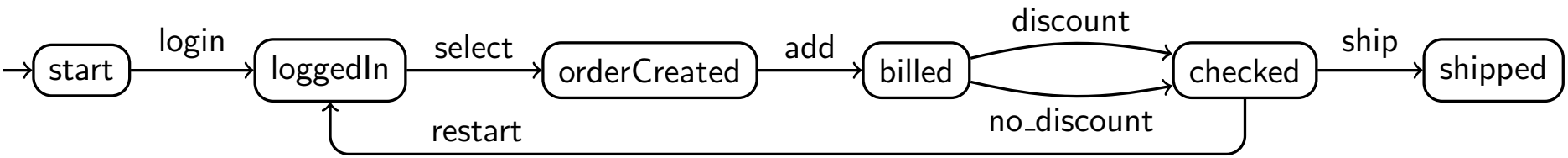
Tools for FOLTL verification

linDMT

Typology	Time horizon	First-order domains	Primed variables	Quantifiers
model checker	finite	unbounded	✓	×

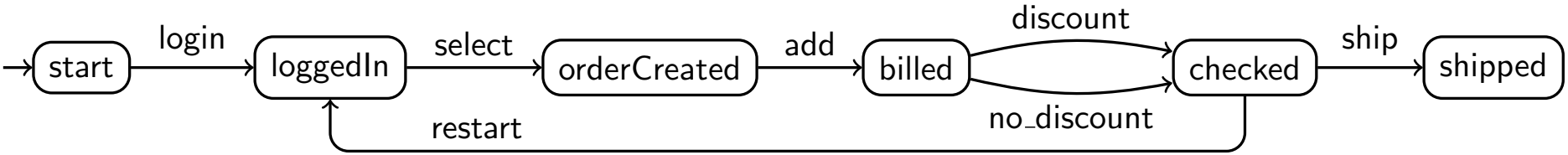
- Encodes ϕ as an automaton and the model as a constraint graph to extract a witness
- Used for data-aware processes and database systems

Webshop example



login:	$\mathsf{cust}(c^w, a^w, v^w)$	select:	$\bigwedge_{i=1}^{5} \mathsf{id}(p_i^w)$
add:	$t^w = \sum_{i=1}^{5} f_{\mathsf{price}}(p_i^r)$	discount:	$v^r \wedge t^w = t^r - \frac{1}{5} t^r$
no_discount:	$\neg v^r \wedge t^w = t^r$	ship:	$t^r \leq a^r$
restart:	$\top$		

Webshop example



login:	$\text{cust}(c^w, a^w, v^w)$	select:	$\bigwedge_{i=1}^{5} \text{id}(p_i^w)$
add:	$t^w = \sum_{i=1}^{5} f_{\text{price}}(p_i^r)$	discount:	$v^r \wedge t^w = t^r - \frac{1}{5} t^r$
no_discount:	$\neg v^r \wedge t^w = t^r$	ship:	$t^r \leq a^r$
restart:	$\top$		

Safety property: no order is ever shipped to a non-VIP customer if their account balance is insufficient to cover the order cost

Liveness property: every order, when created, is eventually shipped or restarted

Webshop example

Implementation in BLACK using FOLTL

```
G(wnext(a) = a) &
G((wnext(p1) = p1) & (wnext(p2) = p2) & (wnext(p3) = p3) & (wnext(p4) = p4) & (wnext(p5) = p5)) &

...

G((start & cust(next(c), next(a), next(v))) -> (X loggedIn & (next(t) = t))) &
G((loggedIn & id(next(p1)) & id(next(p2)) & id(next(p3))) -> (X orderCreated) & (next(t) = t)) &
G(orderCreated -> (X billed & (next(t) = price(p1) + price(p2) + price(p3) + price(p4) + price(p5))))
G((billed & v) -> (X checked & (next(t) = t * 0.80))) &
G((billed & !v) -> (X checked & (next(t) = t))) &
G((checked & (t > a)) -> (X loggedIn & (next(t) = 0.00))) &
G((checked & (t <= a)) -> (X shipped & (next(t) = t))) &

(t = 0.00) &
start & F(shipped & !v & (a < (price(p1) + price(p2) + price(p3) + price(p4) + price(p5))))
```

```
$ black solve -d Real --semi-decision webshop.foltl
```

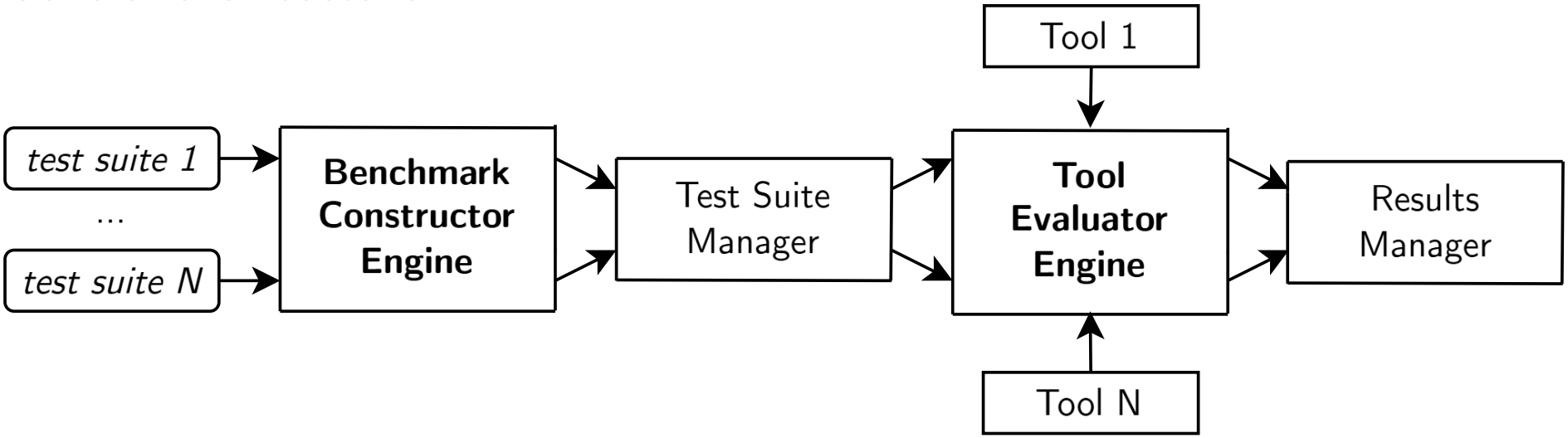
Implementation in Alloy

```
enum State { Start, LoggedIn, OrderCreated, Billed, Checked, Shipped }
one sig Webshop { var state: one State, var customer: lone Customer }
sig Customer { account: one Account, var order: lone Order }
sig VIPCustomer in Customer { }
sig Account { var balance: Int }
sig Order {     var products: set Product, var cost: Int }
sig Product { price: Int }

fact init {
        always transition
        Webshop.state = Start and no customer and no order and no products and Order.cost = 0
        all a: Account | a.balance >= 0 }

pred select[c: Customer, o: Order, p1: Product, p2: Product, p3: Product, p4: Product, p5: Product] {
        Webshop.state = LoggedIn and Webshop.state' = OrderCreated
        Webshop.customer = c and c.order = none and c.order' = o
        o.products = none and o.products' = p1 + p2 + p3 + p4 + p5 and o.cost = 0
        customer' = customer and balance' = balance and cost' = cost }

...

pred transition {
        (some c: Customer, a: Account | login[c, a]) or
        (some c: Customer, o: Order, disj p1, p2, p3, p4, p5: Product | select[c, o, p1, p2, p3, p4, p5])
        (some c: Customer, o: Order, disj p1, p2, p3, p4, p5: Product | add[c, o, p1, p2, p3, p4, p5]) or
        (some c: Customer, o: Order | discount[c, o]) or
        (some c: Customer, o: Order | no_discount[c, o]) or
        (some c: Customer, o: Order | restart[c, o]) or
        (some c: Customer, o: Order | ship[c, o]) }

pred orderEventuallyShippedOrRestarted {
        always (Webshop.state = OrderCreated implies
                        eventually (Webshop.state = Shipped or Webshop.state = LoggedIn)) }
run orderEventuallyShippedOrRestarted for 1 Customer, 1 Account, 1 Order, 5 Product, 8 Int
```

Proposal

Main goal: provide a comprehensive and standardized evaluation of formal verification tools for FOLTL

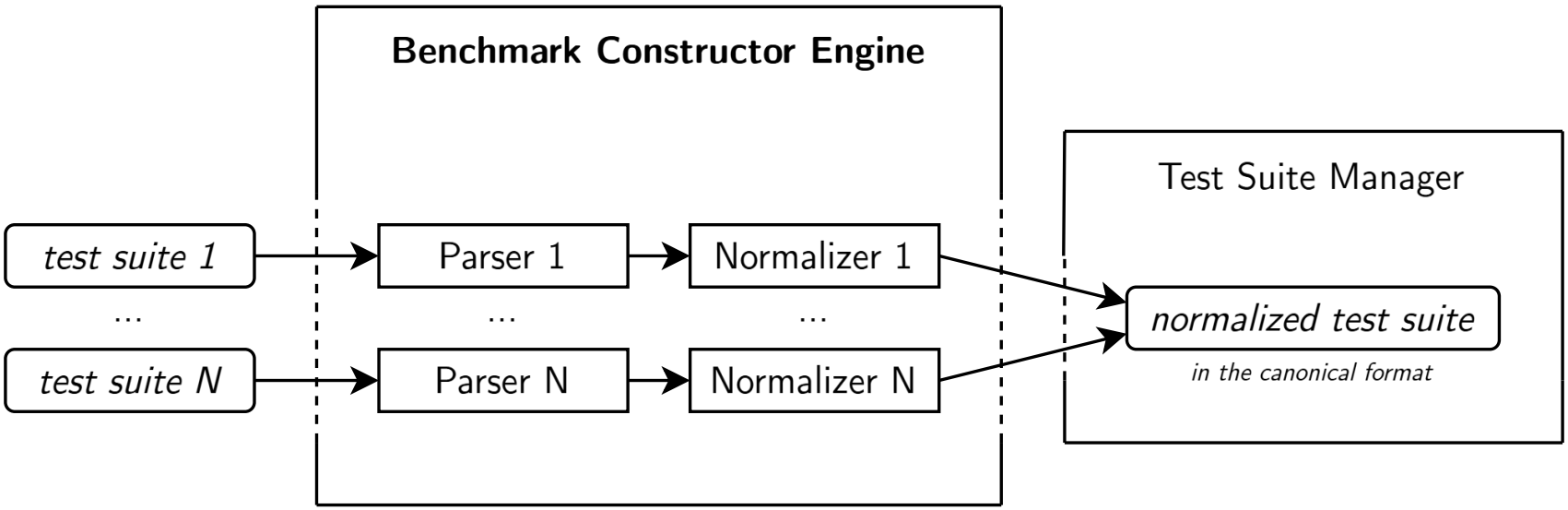
General architecture:



Proposal

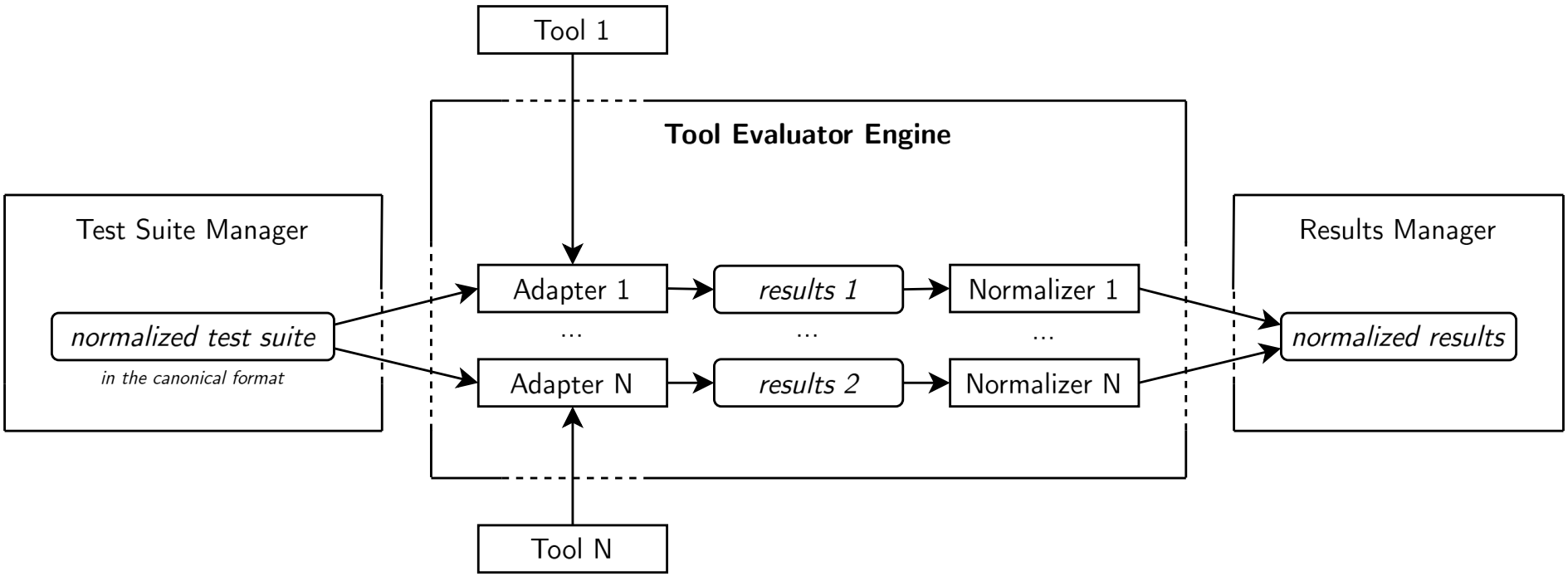
Benchmark Contructor Engine

- Collection of formulas
- Definition of a canonical format
- Benchmark construction



Tool Evaluator Engine

- Development of a generic executor
- Metrics measurement
- Data analysis



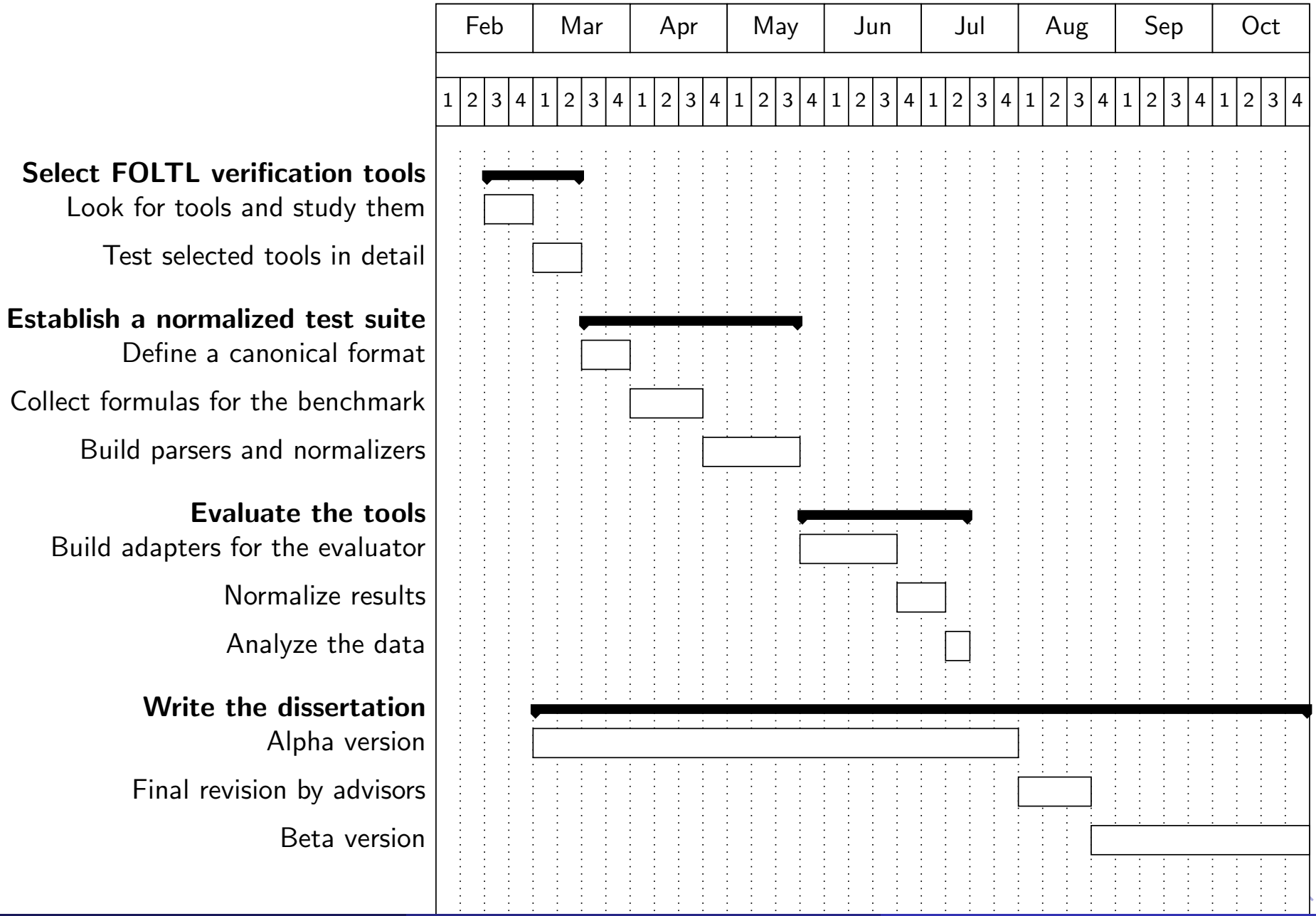
Proposal

Portfolio

The evaluator could be extended into an intelligent interface for tool selection:

- using insights from past evaluations to select the best tool for each formula
- serving as a unified front-end for different FOLTL tools
- further abstracting the tools

Schedule


Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
Select FOLTL verification tools
Look for tools and study them
Test selected tools in detail
Establish a normalized test suite
Define a canonical format
Collect formulas for the benchmark
Build parsers and normalizers
Evaluate the tools
Build adapters for the evaluator
Normalize results
Analyze the data
Write the dissertation
Alpha version
Final revision by advisors
Beta version

Conclusion

Contribution

- A comprehensive benchmark of formulas for assessing the most relevant FOLTL verification tools
- An empirical study to compare the expressiveness, effectiveness, and performance of these tools