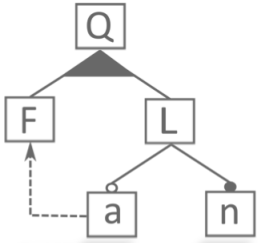
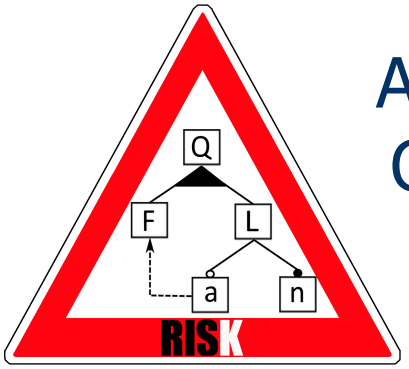


QFLan: A tool for the Quantitative Analysis of Highly Reconfigurable Systems



A Software Engineering Approach to Quantitative Security Risk Modeling and Analysis using QFLan

Andrea Vandin
Sant'Anna School of Advanced Studies Pisa, Italy
DTU Technical University of Denmark

Maurice H. ter Beek
ISTI CNR Pisa, Italy

Axel Legay
UCLouvain, Belgium

Alberto Lluch Lafuente
DTU, Denmark

Tools for Stochastic Modelling and Evaluation
(performance, dependability, security and verification) 12/11/2021

References

[COSE21] Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, Quantitative Security Risk Modeling and Analysis with RisQFLan. Computers & Security (COSE), 2021.

[TSE18] Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, A framework for quantitative modeling and analysis of highly (re)configurable systems, IEEE Transactions on Software Engineering (TSE), 2018.

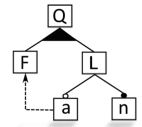
[FM18] Andrea Vandin, Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems.

[ISOLA16] Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, Statistical Model Checking for Product Lines.

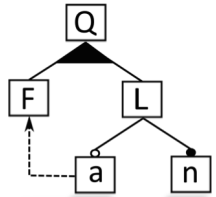
[SPLC15] Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, Statistical Analysis of Probabilistic Models of Software Product Lines with Quantitative Constraints.

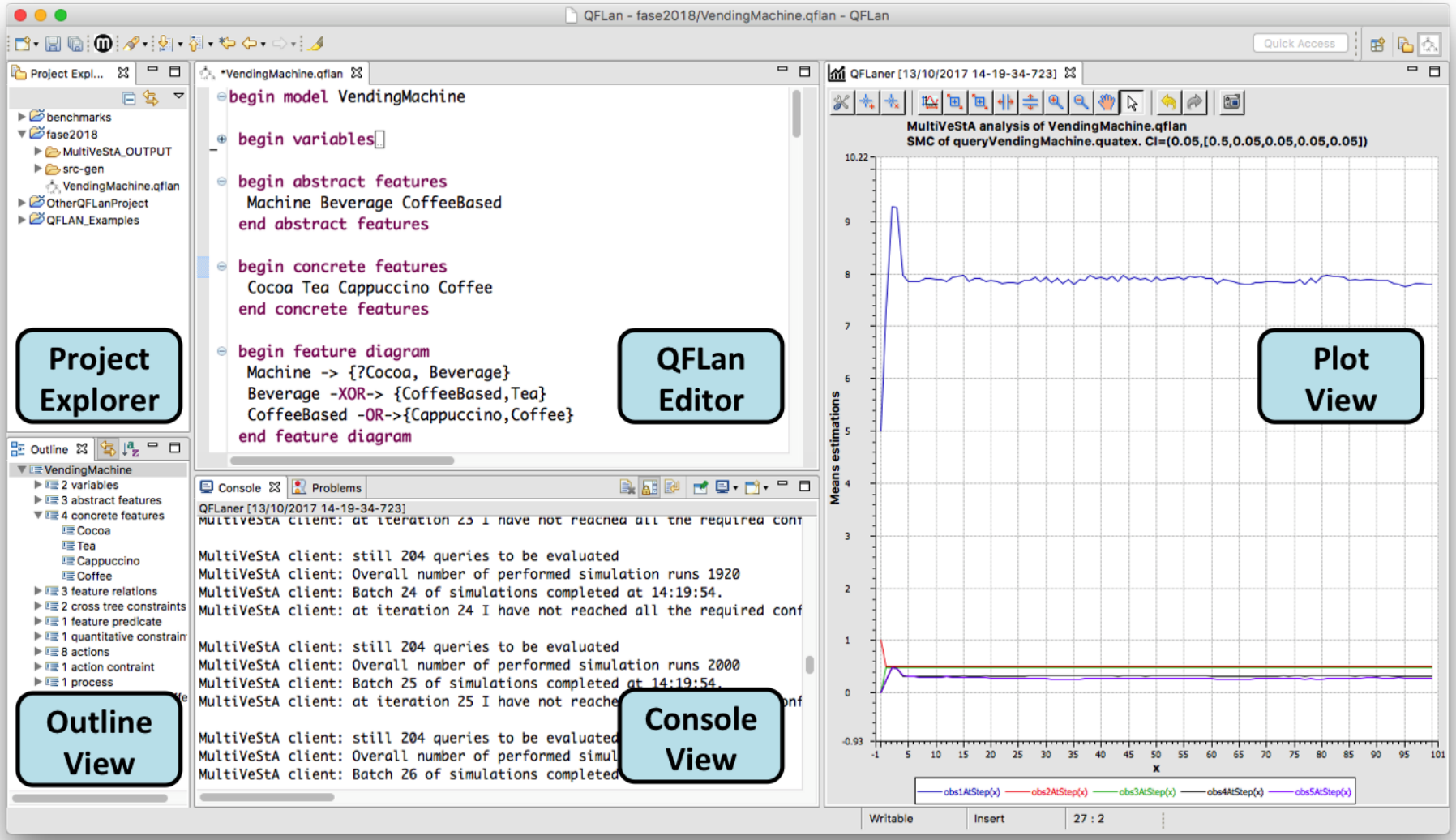
[FMSPLE15] Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, Quantitative Analysis of Probabilistic Models of Software Product Lines with Statistical Model Checking.

bit.ly/RisQFLan

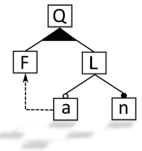


Presented in [FM'18][TSE'18]
Prototypes in [FMSPLE'15][SPLC'15][ISOLA'16]

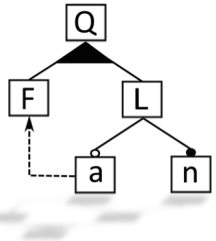


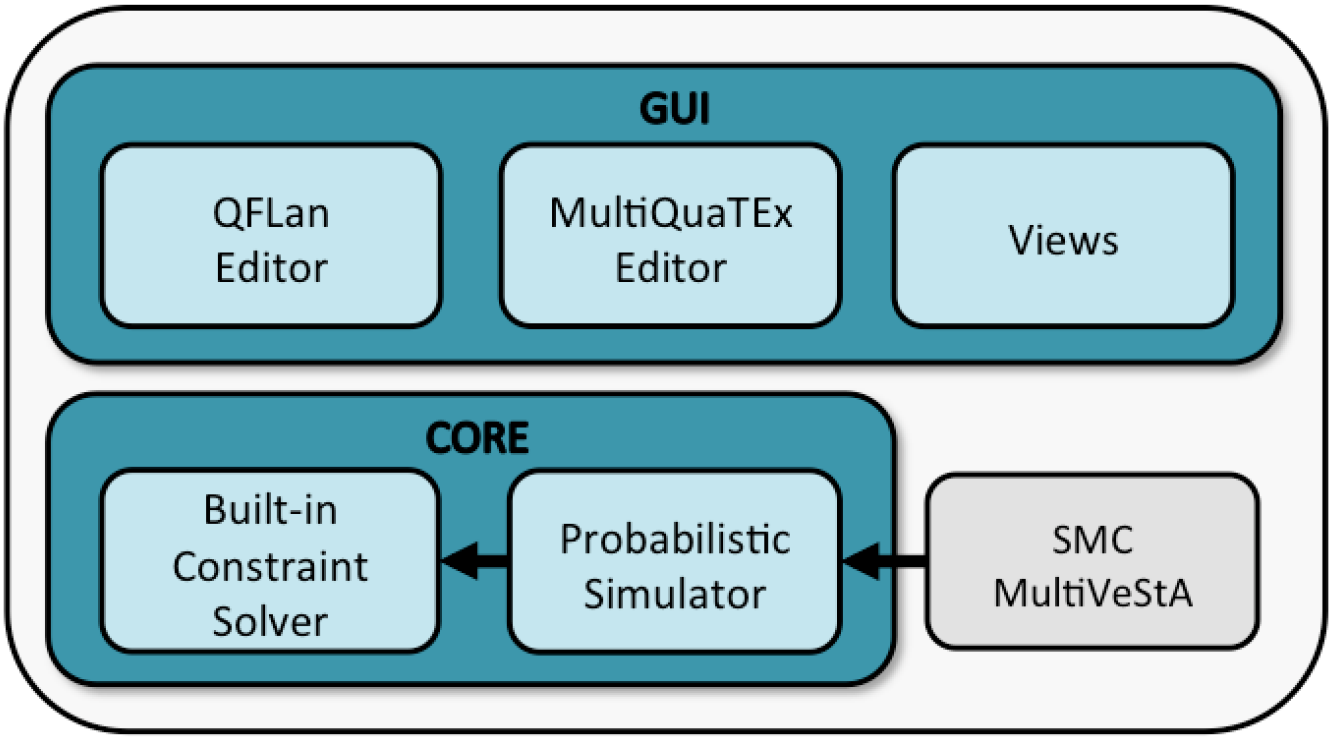


https://github.com/qflanTeam/QFLan/

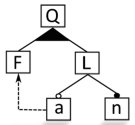


Presented in [FM'18][TSE'18]
Prototypes in [FMSPLE'15][SPLC'15][ISOLA'16]



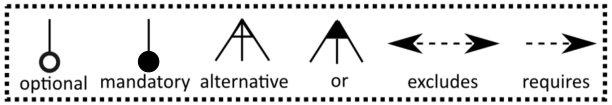


https://github.com/qflanTeam/QFLan/

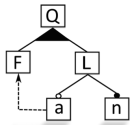


A simple vending machine product line

The feature model

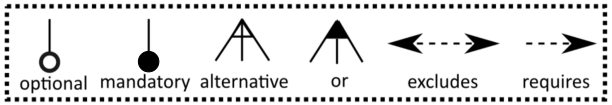


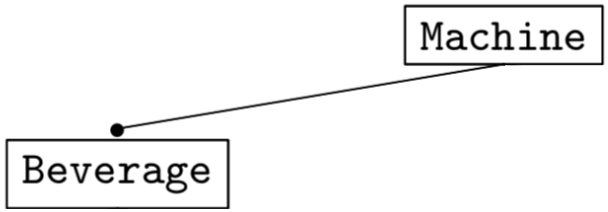
Machine

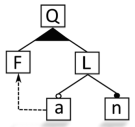


A simple vending machine product line

The feature model

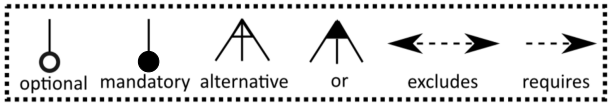


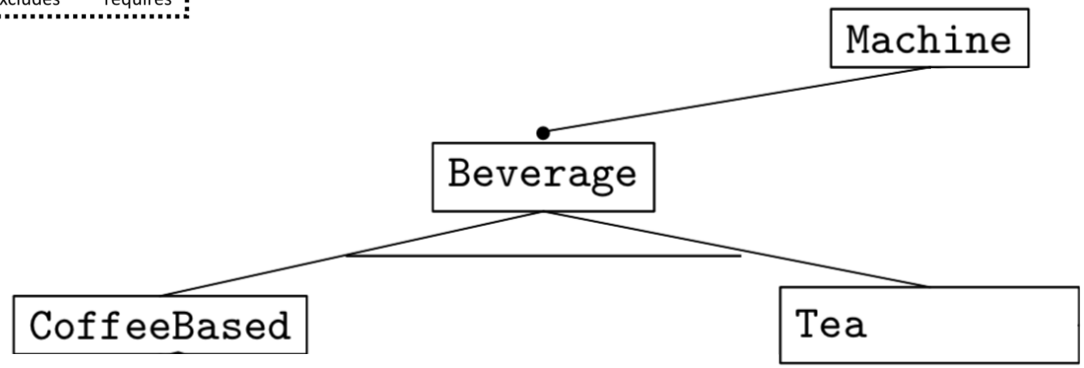


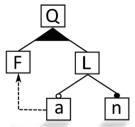


A simple vending machine product line

The feature model

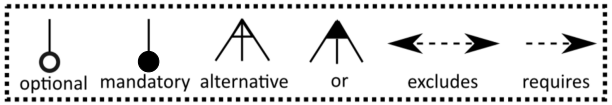


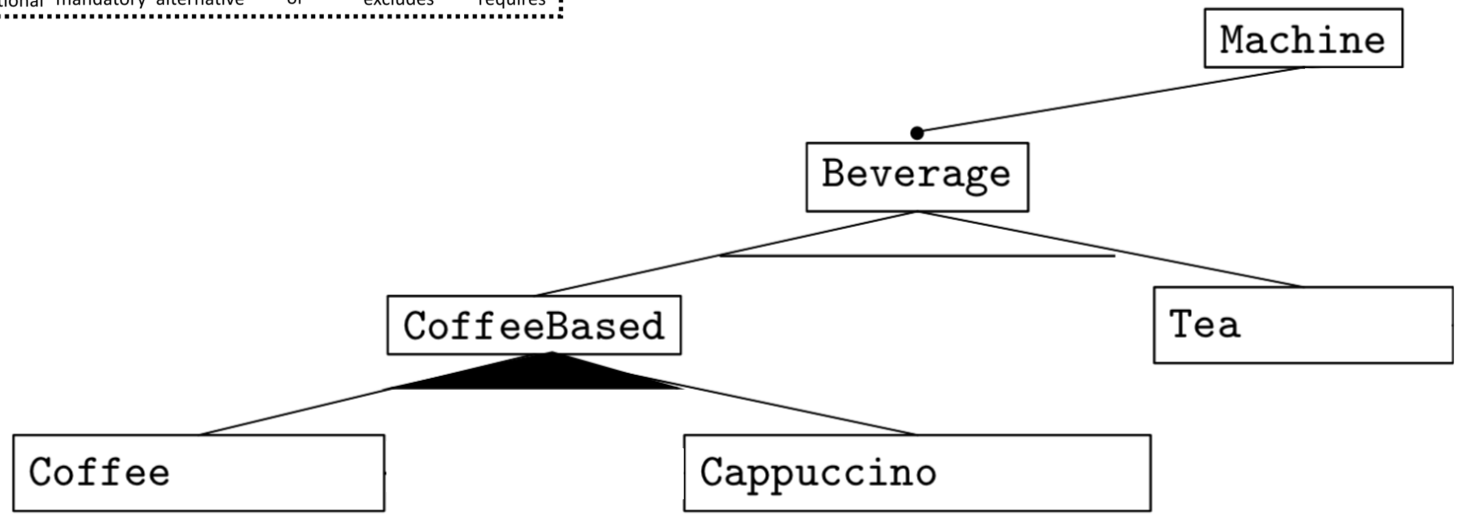


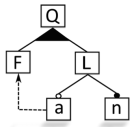


A simple vending machine product line

The feature model

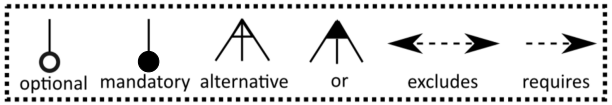


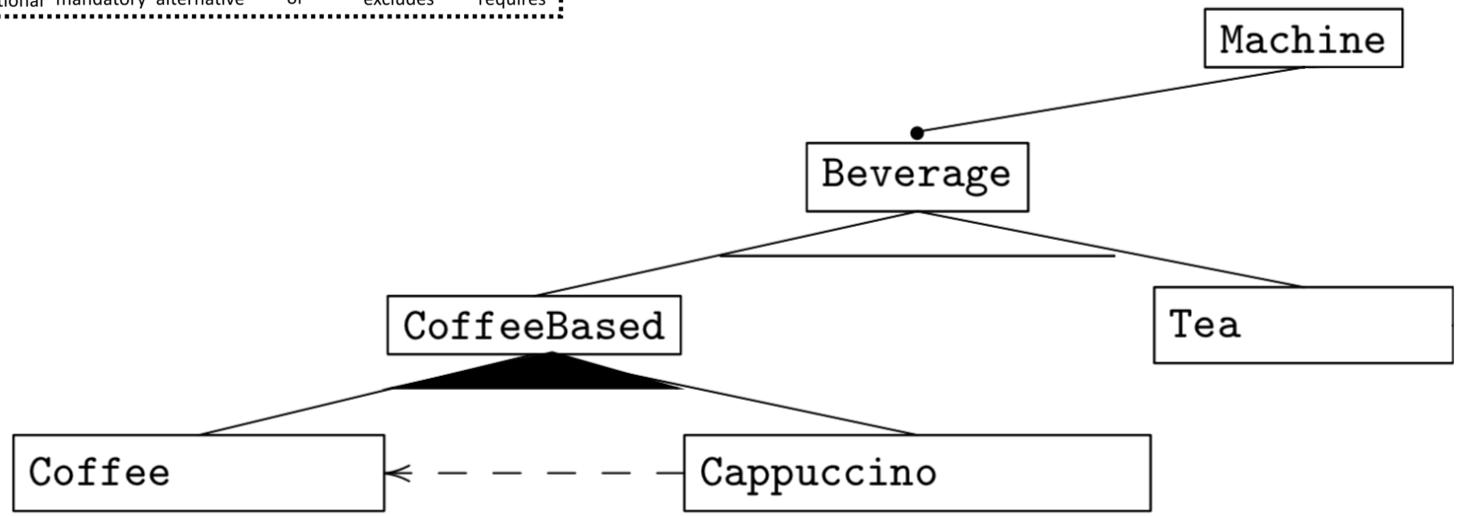




A simple vending machine product line

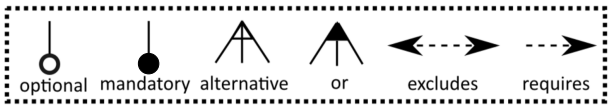
The feature model

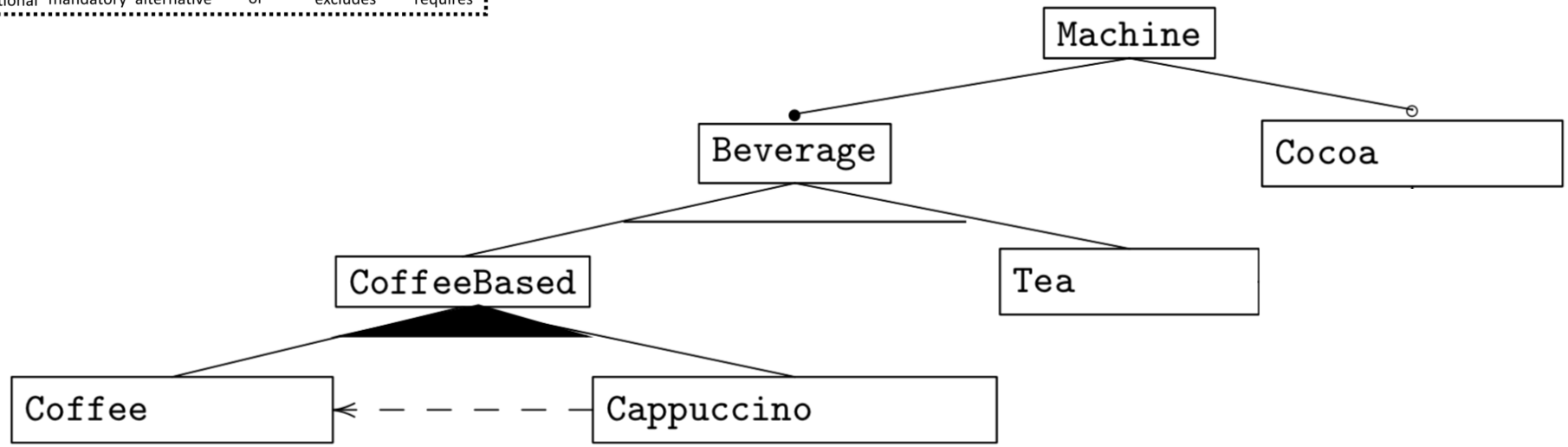




A simple vending machine product line

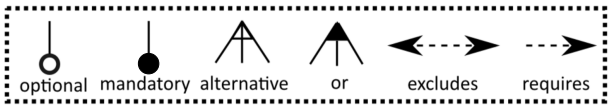
The feature model

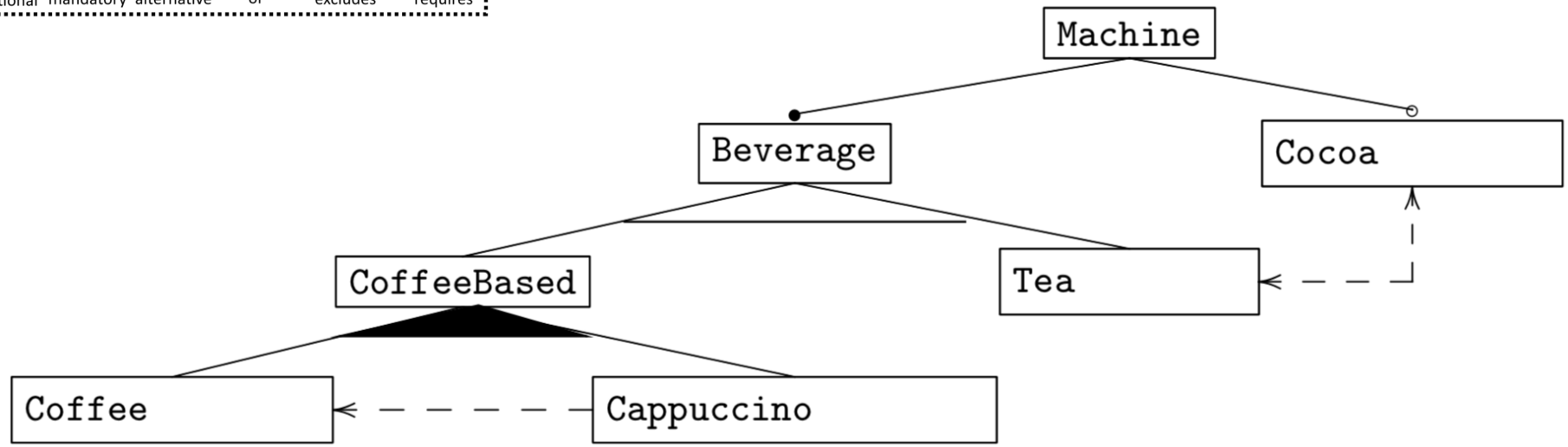




A simple vending machine product line

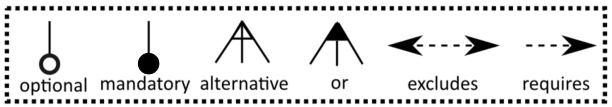
The feature model

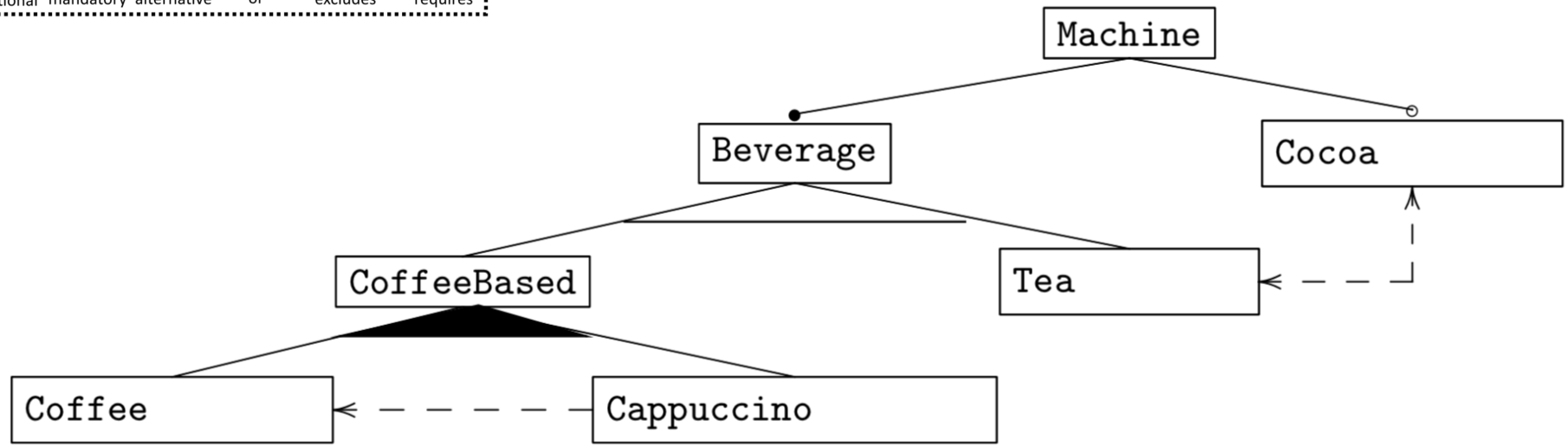




A simple vending machine product line

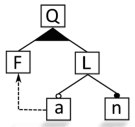
The feature model: Abstract & Concrete Features





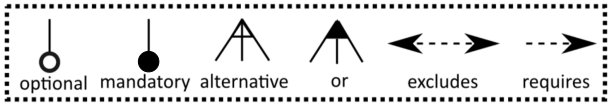

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

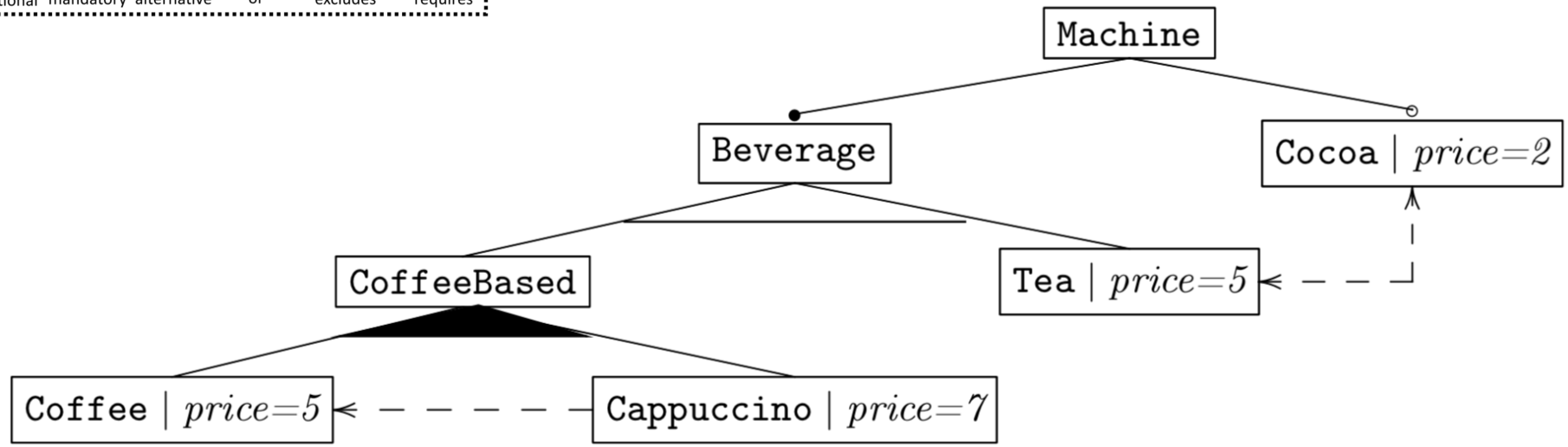
```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
```



A simple vending machine product line

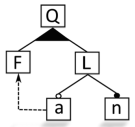
The feature model: Abstract & Concrete Features





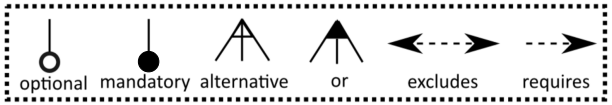

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

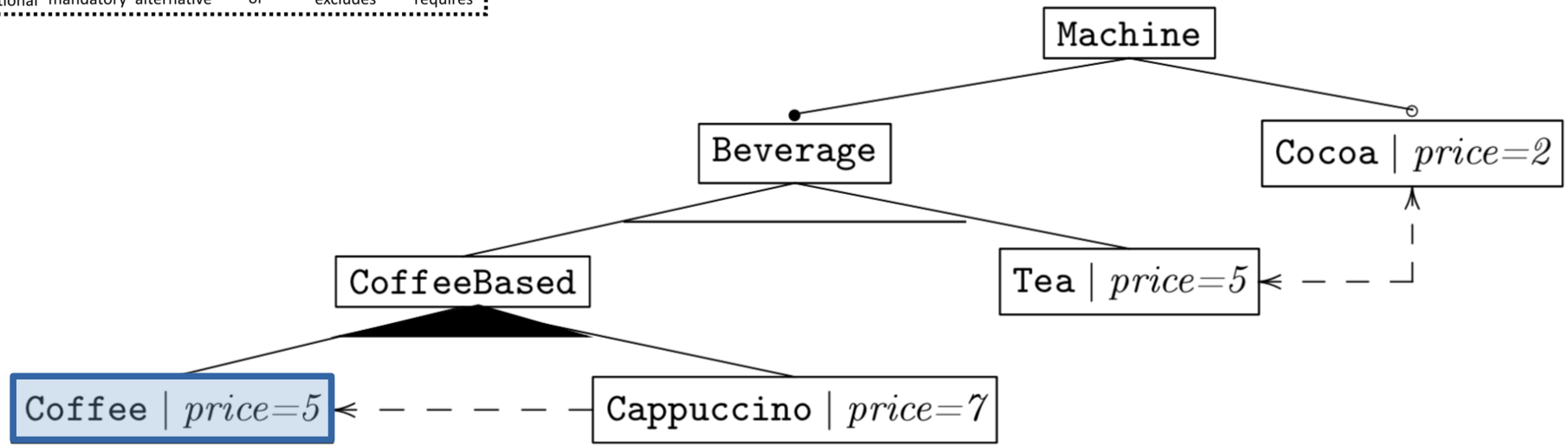
```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
```



A simple vending machine product line

The feature model: Abstract & Concrete Features



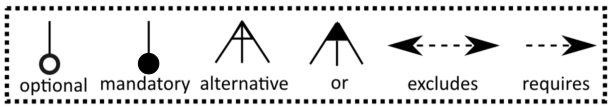



```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
```

A simple vending machine product line

The feature model: Abstract & Concrete Features

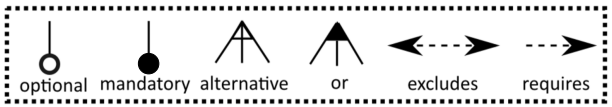



```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
```

A simple vending machine product line

The feature model: Abstract & Concrete Features

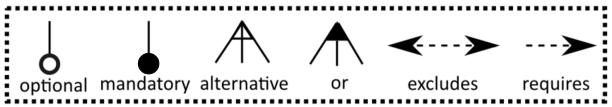



```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
```

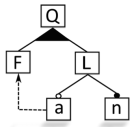
A simple vending machine product line

The feature model: Abstract & Concrete Features



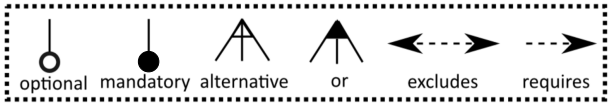

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

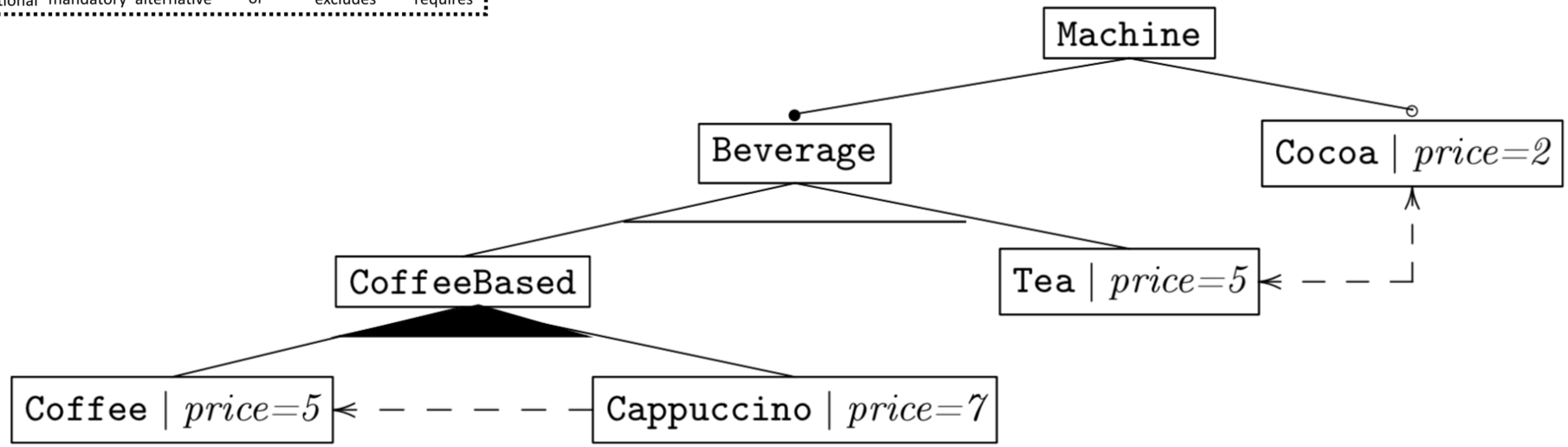
```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
```



A simple vending machine product line

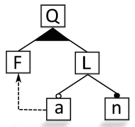
The feature model: Quantitative constraints





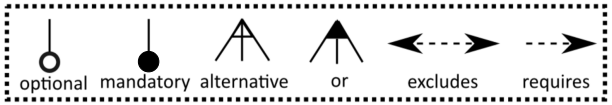

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

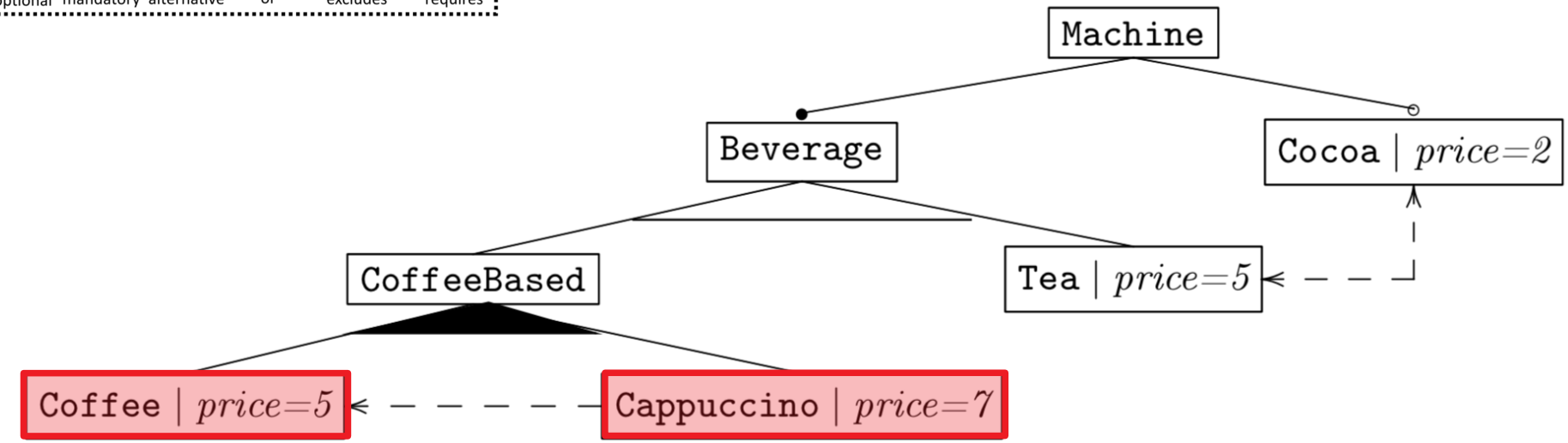
```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```



A simple vending machine product line

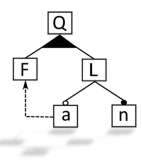
The feature model: Quantitative constraints





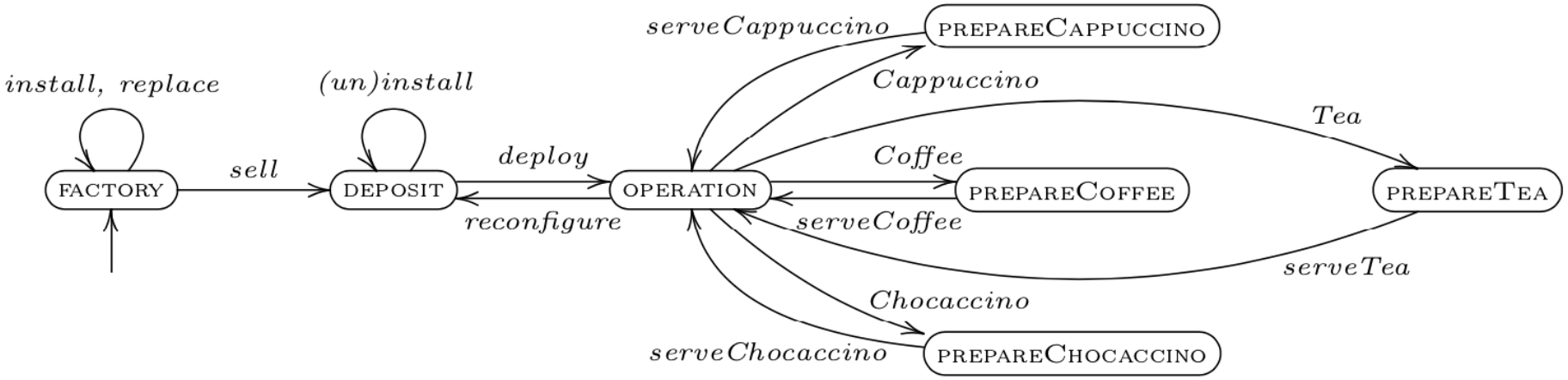

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

begin concrete features
 Cocoa Tea Cappuccino Coffee
end concrete features

begin feature diagram
 Machine -> {?Cocoa, Beverage}
 Beverage -XOR-> {CoffeeBased,Tea}
 CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

begin cross-tree constraints
 Cappuccino requires Coffee
 Tea excludes Cocoa
end cross-tree constraints
```

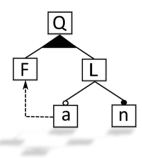
```
begin feature predicates
 price= { Cappuccino = 7, Coffee = 5,
          Cocoa = 2, Tea = 5 }
end feature predicates
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```



A simple vending machine product line

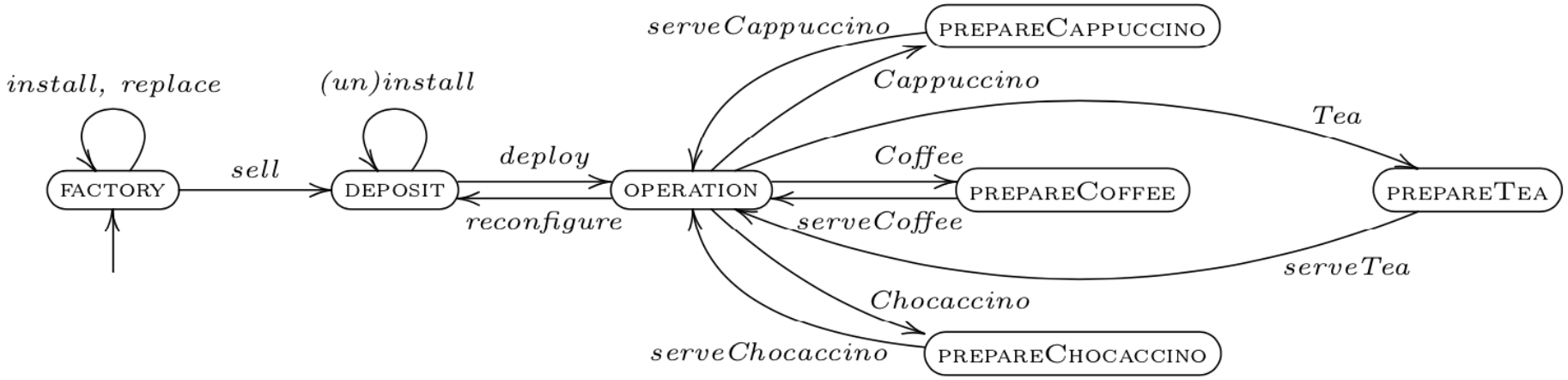
Behaviour: actions and action constraints





A simple vending machine product line

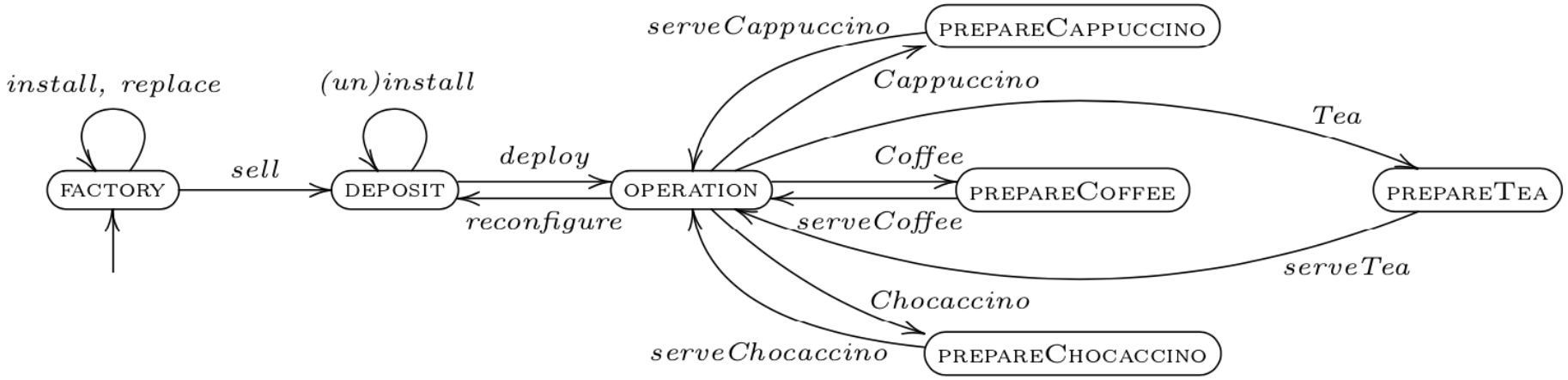
Behaviour: actions and action constraints




```
begin actions
 sell deploy reconfigure
 chocaccino
 serveCoffee serveCappuccino
 serveChocaccino serveTea
end actions

begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

A simple vending machine product line

Behaviour: transitions



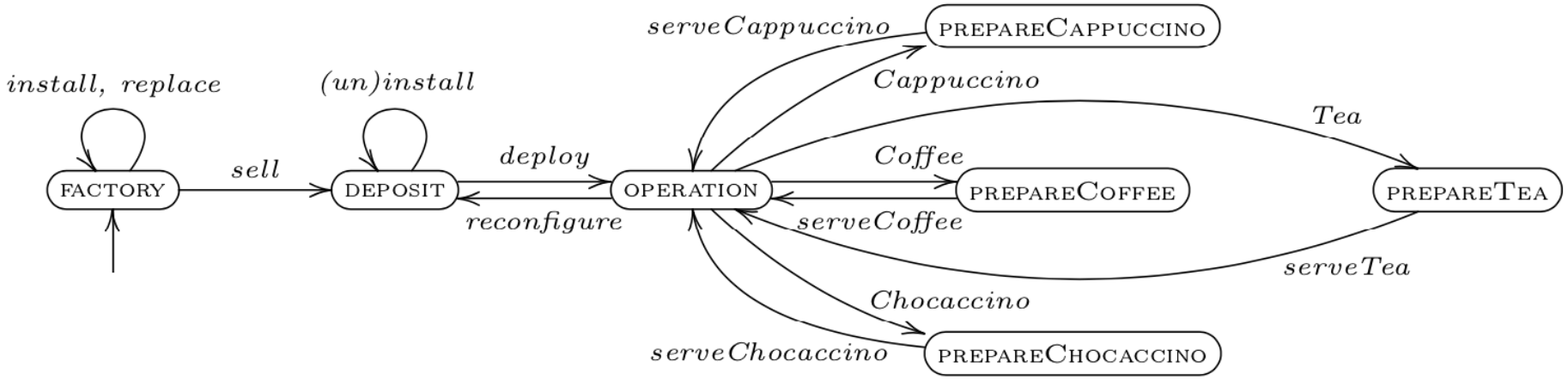

```
begin processes diagram
 begin process dynamics

 states = factory , deposit , operating , prepareCoffee ,
          prepareCappuccino, prepareTea , prepareChocaccino
```

```
begin actions
 sell deploy reconfigure
 chocaccino
 serveCoffee serveCappuccino
 serveChocaccino serveTea
end actions

begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

,

A simple vending machine product line

Behaviour: transitions

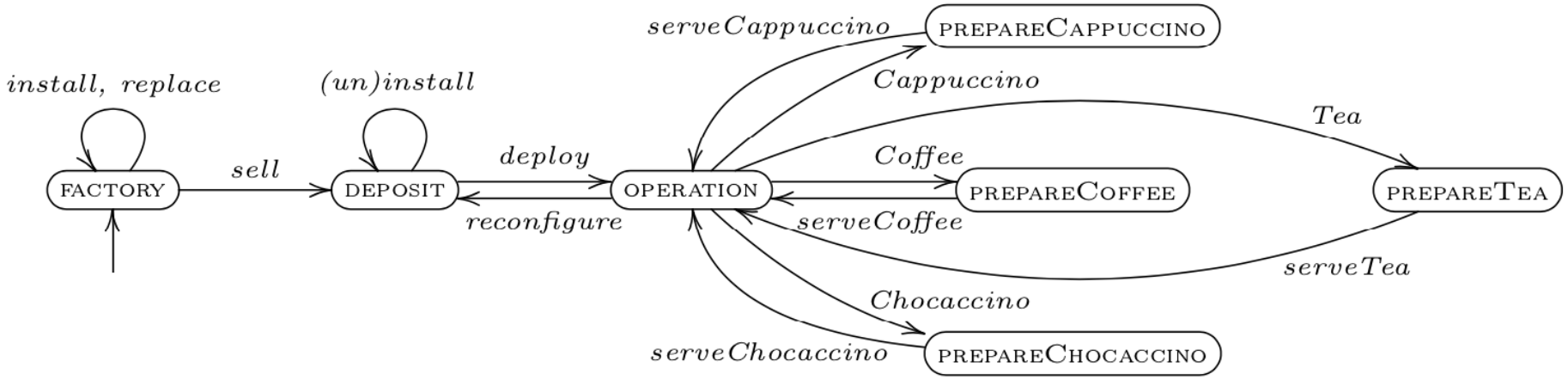



```
begin actions
 sell deploy reconfigure
 chocaccino
 serveCoffee serveCappuccino
 serveChocaccino serveTea
end actions

begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

```
begin processes diagram
 begin process dynamics

 states = factory , deposit , operating , prepareCoffee ,
          prepareCappuccino, prepareTea , prepareChocaccino

 transitions =
   //Factory
   factory -(replace(Coffee,Tea),20)->factory,
   factory -(install(Cocoa),10)->factory,
   factory -(install(Cappuccino),10)->factory,
   factory -(sell,1,{sold=1})-> deposit,

   //Deposit
   deposit -(install(Cappuccino),2.0)->deposit,
   deposit -(uninstall(Cappuccino),2.0)->deposit,
   deposit -(install(Cocoa),2.0)->deposit,
   deposit -(uninstall(Cocoa),2.0)->deposit,
   deposit -(deploy,2,{deploys=deploys+1})-> operating ,
```

A simple vending machine product line

Behaviour: transitions

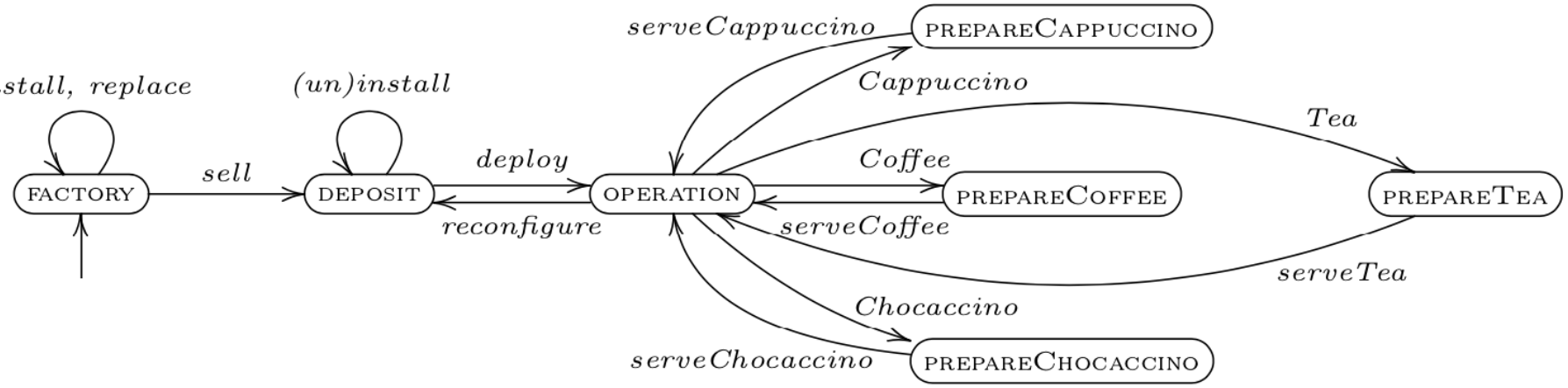



```
begin variables
 sold = 0
 deploys = 0
end variables

begin actions
 sell deploy reconfigure
 chocaccino
 serveCoffee serveCappuccino
 serveChocaccino serveTea
end actions

begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

```
begin processes diagram
 begin process dynamics

 states = factory , deposit , operating , prepareCoffee ,
          prepareCappuccino, prepareTea , prepareChocaccino

 transitions =
   //Factory
   factory -(replace(Coffee,Tea),20)->factory,
   factory -(install(Cocoa),10)->factory,
   factory -(install(Cappuccino),10)->factory,
   factory -(sell,1,{sold=1})-> deposit,

   //Deposit
   deposit -(install(Cappuccino),2.0)->deposit,
   deposit -(uninstall(Cappuccino),2.0)->deposit,
   deposit -(install(Cocoa),2.0)->deposit,
   deposit -(uninstall(Cocoa),2.0)->deposit,
   deposit -(deploy,2,{deploys=deploys+1})-> operating ,
```

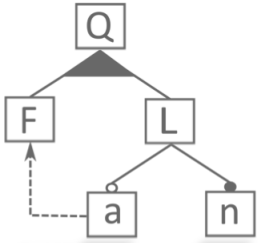
A simple vending machine product line

Behaviour: transitions

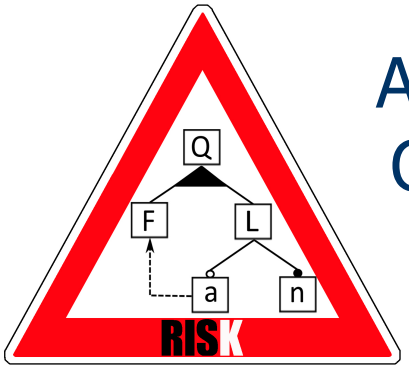



```
begin variables
 sold = 0
 deploys = 0
end variables

begin actions
 sell deploy reconfigure
 chocaccino
 serveCoffee serveCappuccino
 serveChocaccino serveTea
end actions

begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

```
begin processes diagram
 begin process dynamics

 states = factory , deposit , operating , prepareCoffee ,
          prepareCappuccino, prepareTea , prepareChocaccino

 transitions =
   //Factory
   factory -(replace(Coffee,Tea),20)->factory,
   factory -(install(Cocoa),10)->factory,
   factory -(install(Cappuccino),10)->factory,
   factory -(sell,1,{sold=1})-> deposit,

   //Deposit
   deposit -(install(Cappuccino),2.0)->deposit,
   deposit -(uninstall(Cappuccino),2.0)->deposit,
   deposit -(install(Cocoa),2.0)->deposit,
   deposit -(uninstall(Cocoa),2.0)->deposit,
   deposit -(deploy,2,{deploys=deploys+1})-> operating

   //Operating
       //Coffee
   operating -(Coffee,3)-> prepareCoffee,
   prepareCoffee -(serveCoffee,1) -> operating,
       //Cappuccino
   operating -(Cappuccino,3)-> prepareCappuccino,
   prepareCappuccino -(serveCappuccino,1) -> operating,
       //Chocaccino
   operating -(chocaccino,2)-> prepareChocaccino,
   prepareChocaccino -(serveChocaccino,1) -> operating,
       //Tea
   operating -(Tea,3)-> prepareTea,
   prepareCappuccino -(serveTea,1) -> operating,

   operating -(reconfigure,1) -> deposit
 end process
end processes diagram
```



QFLan: A tool for the Quantitative Analysis of Highly Reconfigurable Systems



A Software Engineering Approach to Quantitative Security Risk Modeling and Analysis using QFLan

Andrea Vandin
Sant'Anna School of Advanced Studies Pisa, Italy
DTU Technical University of Denmark

Maurice H. ter Beek
ISTI CNR Pisa, Italy

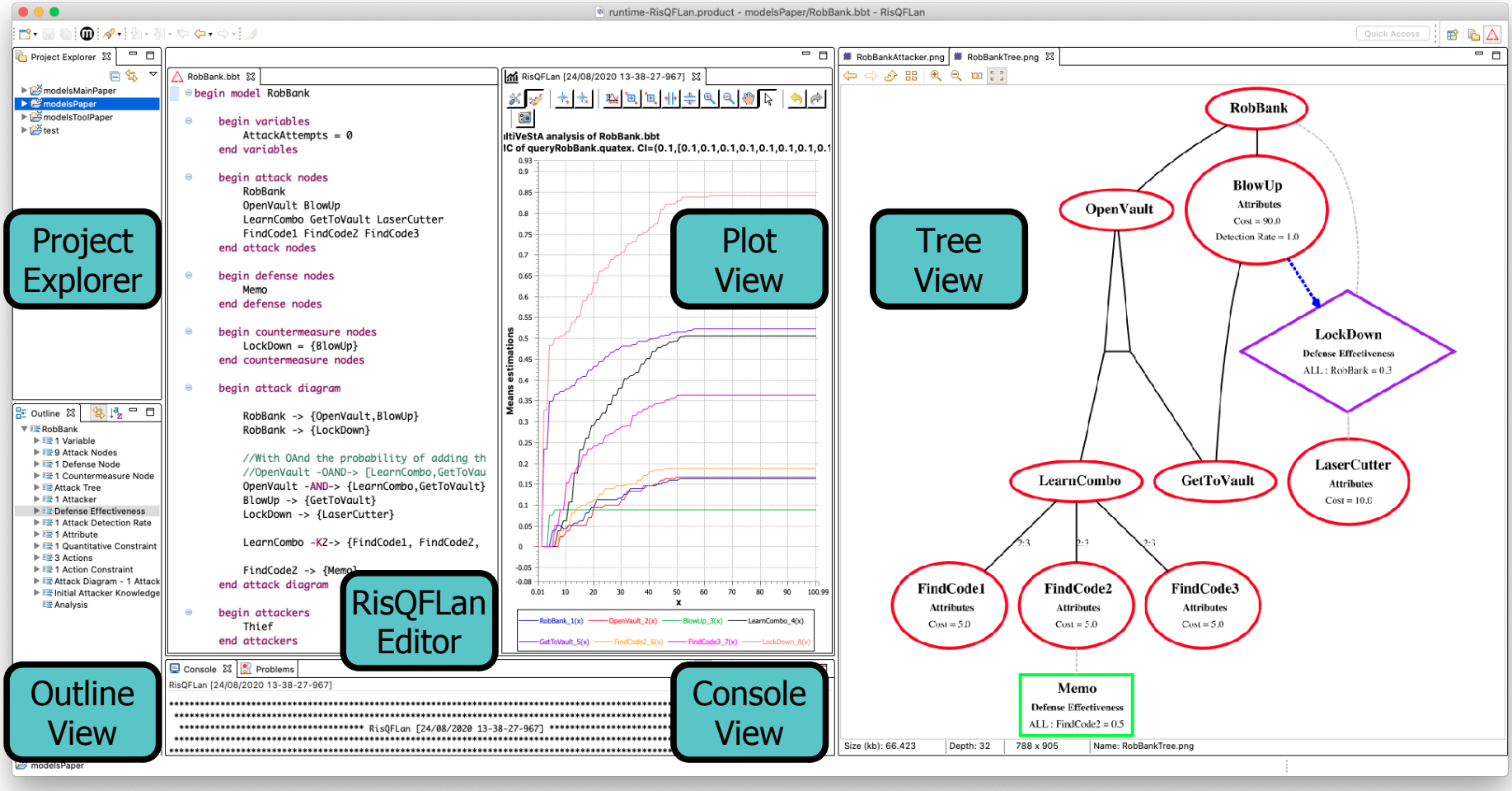
Axel Legay
UCLouvain, Belgium

Alberto Lluch Lafuente
DTU, Denmark

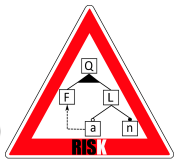
PhD program in Smart Computing, Online, 16/07/2021

A Bank Robbery Scenario in RisQFLan

A screenshot of RisQFLan



bit.ly/RisQFLan



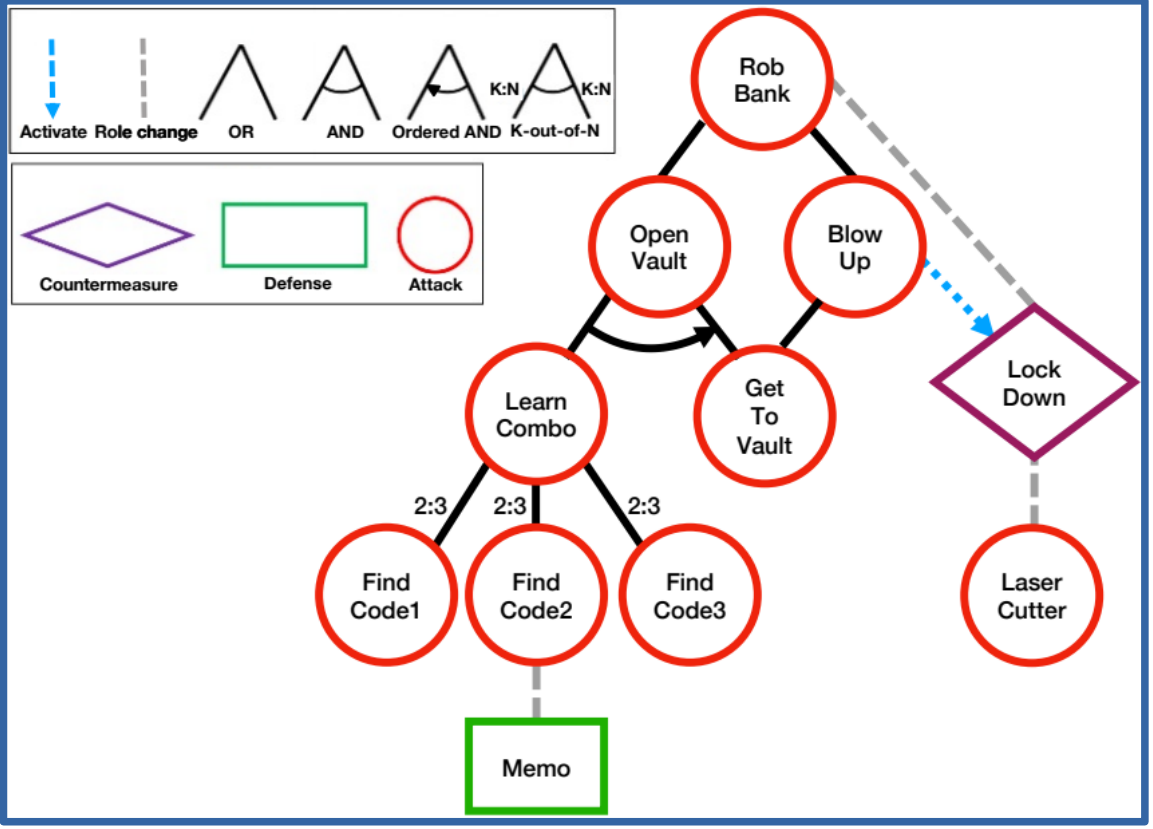
QFLan limitations for Risk Modeling/Analysis

- Not entirely direct encoding of the scenario
  - The extra root node, the extra states to model failures, etc
- We need different types of nodes
  - Attack, defense, countermeasure
- We need richer constructs for building the tree diagram
  - QFLan has: or, requires, excludes
  - Missing *common* constructs: and, o-and, n-out-of-k, activates, inhibits
- Attack attempts might fail
  - The 'install' of an attack node might 'fail'. Failures should be 1st-class citizens
- There is no 'absolute security'
  - *Qualitative* constraints like 'excludes' or 'requires' are too strong
  - Often, failure probabilities are 'scaled' and not zeroed by defense mechanisms
- Exact analysis might be necessary in some scenarios
  - Complement MultiVeStA Statistical MC by PRISM/STORM exact Probabilistic MC

QFLan limitations for Risk Modeling/Analysis

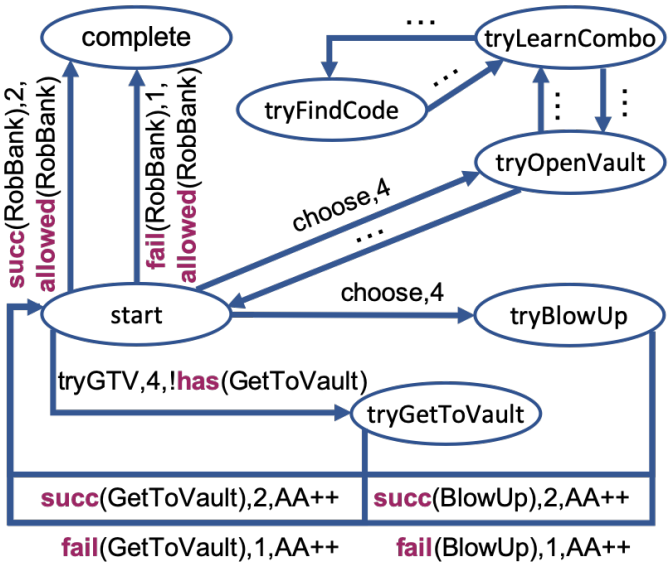
- Not entirely direct encoding of the scenario
  - The extra root node, the extra states to model failures, etc
- We need different types of nodes
  - Attack, defense, counterme
- We need richer constru
  - QFLan has: or, requires, exc
  - Missing *common* constructs
- Attack attempts might f
  - The 'install' of an attack noc
- There is no 'absolute se
  - *Qualitative* constraints like '
  - Often, failure probabilities a
- Exact analysis might be
  - Complement MultiVeStA Sta

We can model scenarios like this



A Bank Robbery Scenario in RisQFLan

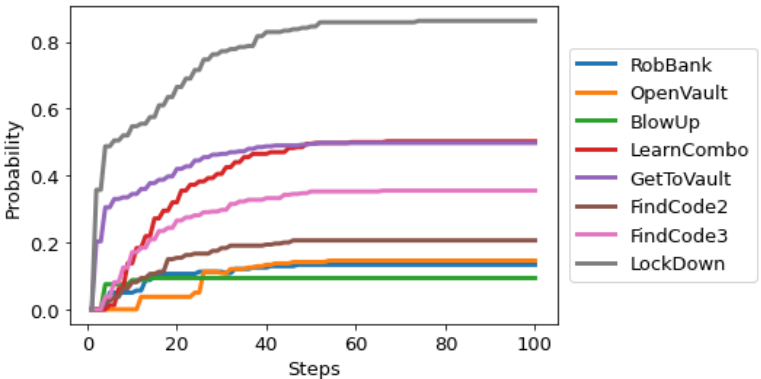
Analysis: SMC with MultiVeStA

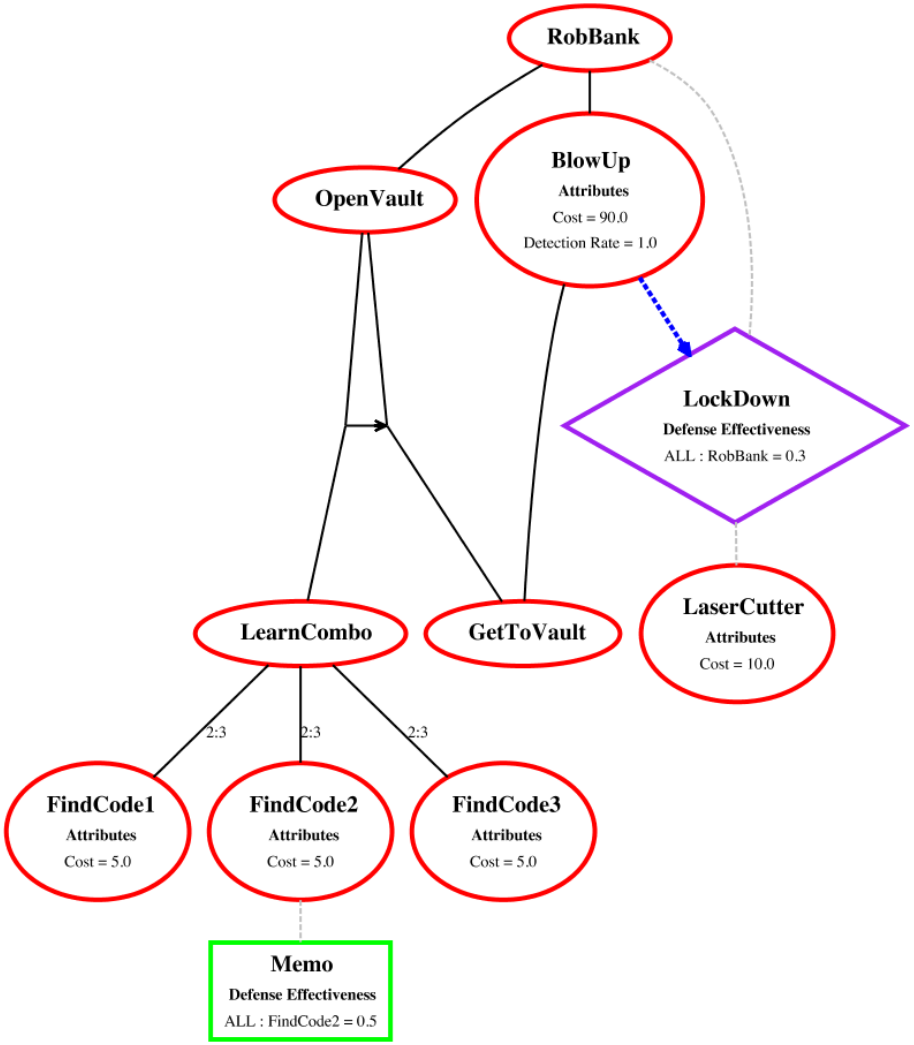


Attacker Behaviour

```
begin analysis
    query = eval from 1 to 100 by 1 :
    { RobBank, OpenVault, BlowUp,
      LearnCombo, GetToVault,
      FindCode2, FindCode3, LockDown }
    default delta = 0.1 alpha = 0.1
    parallelism = 1
end analysis
```

Statistical SMC Analysis



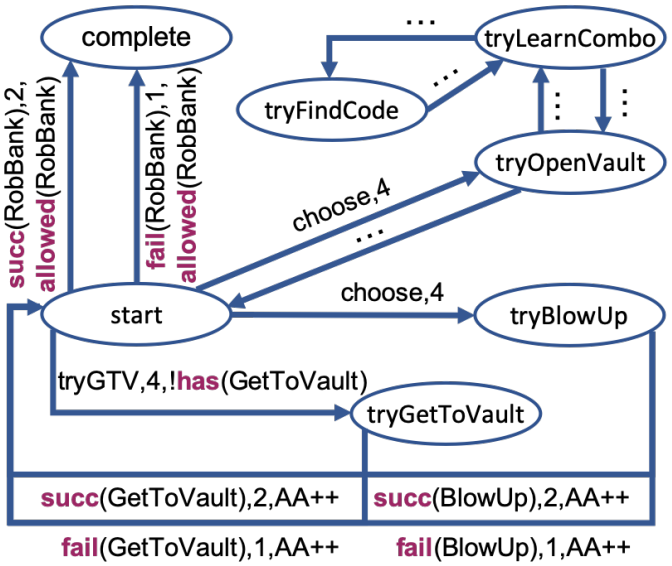


Attack-defense tree

bit.ly/RisQFLan

A Bank Robbery Scenario in RisQFLan

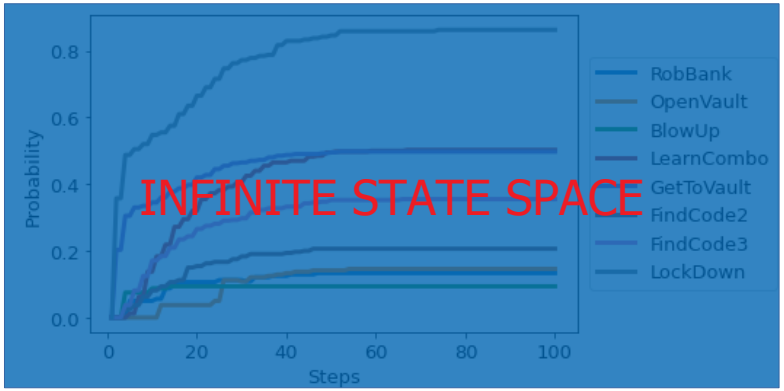
Analysis: PMC with PRISM/STORM

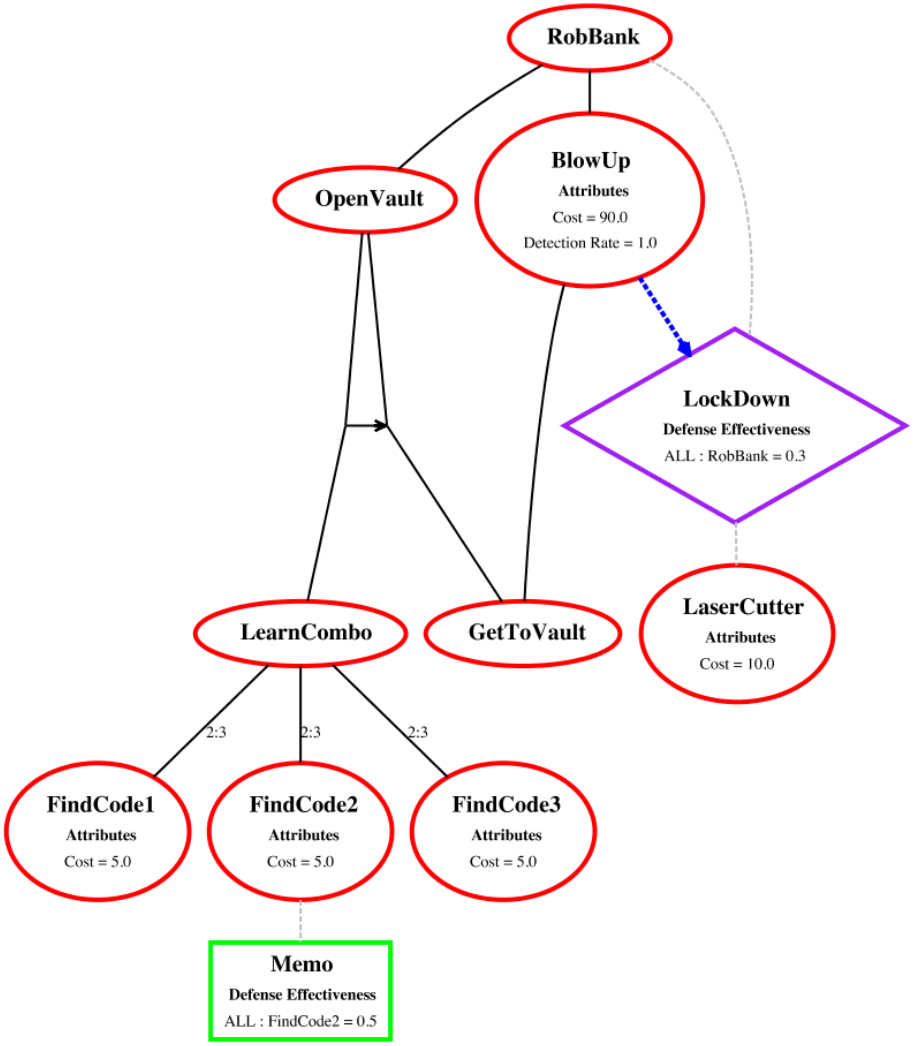


Attacker Behaviour

```
begin exportDTMC
  file ="RobBank.prism"
  label with "Succeeded"
        when has(RobBank)
end exportDTMC
```

Exact
PMC Analysis



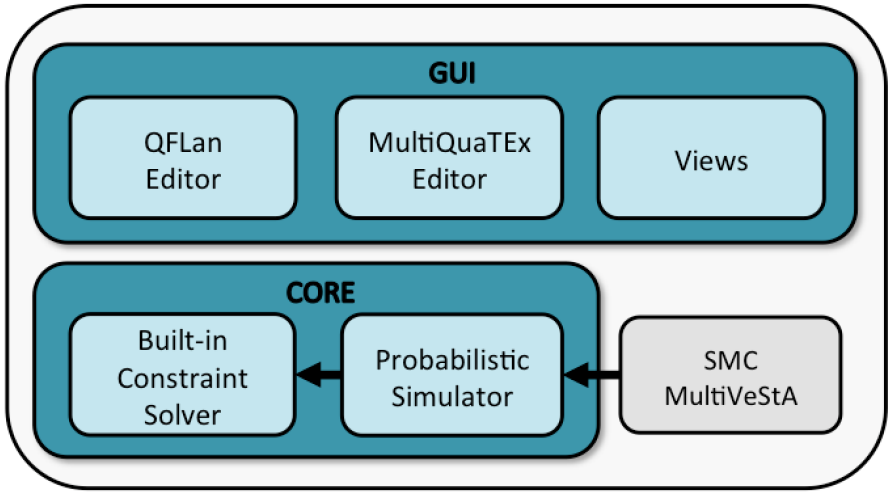


Attack-defense tree

bit.ly/RisQFLan

From QFLan to RisQFLan
Generalizing the QFLan approach

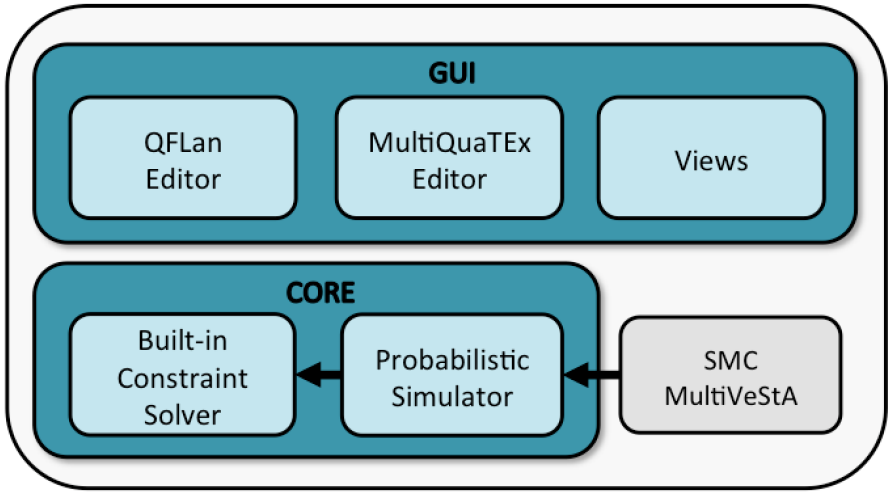
QFLan Architecture [FM'18][TSE'18]



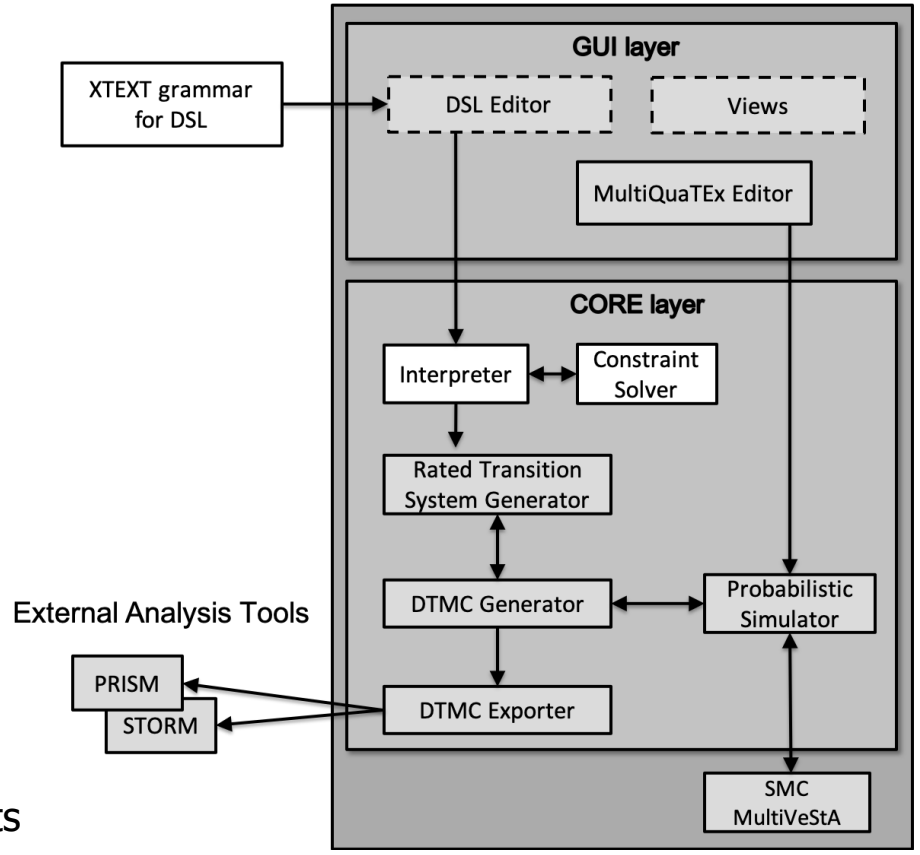
From QFLan to RisQFLan

Generalizing the QFLan approach

QFLan Architecture [FM'18][TSE'18]



Generalized QFLan Architecture [COSE'21]



… Existing domain-independent components

… Automatically generated domain-independent components

… Domain-specific components necessary to instantiate the architecture in a new domain

QUESTIONS?

bit.ly/RisQFLan