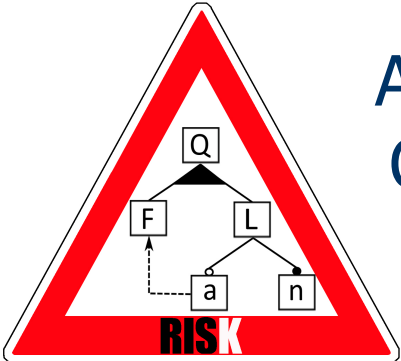


A Software Engineering Approach to Quantitative Security Risk Modeling and Analysis using QFLan



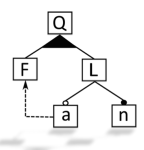
Andrea Vandin

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



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.

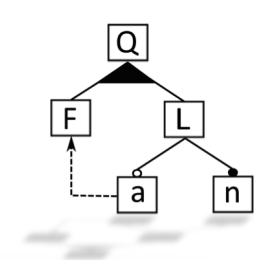
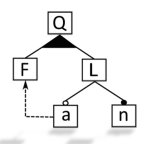
[FM18] Andrea Vandin, Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems.

[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.

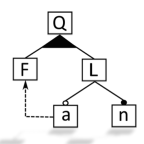
[FMSPLE15] Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, Quantitative Analysis of Probabilistic Models of Software Product Lines with Statistical Model Checking.

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

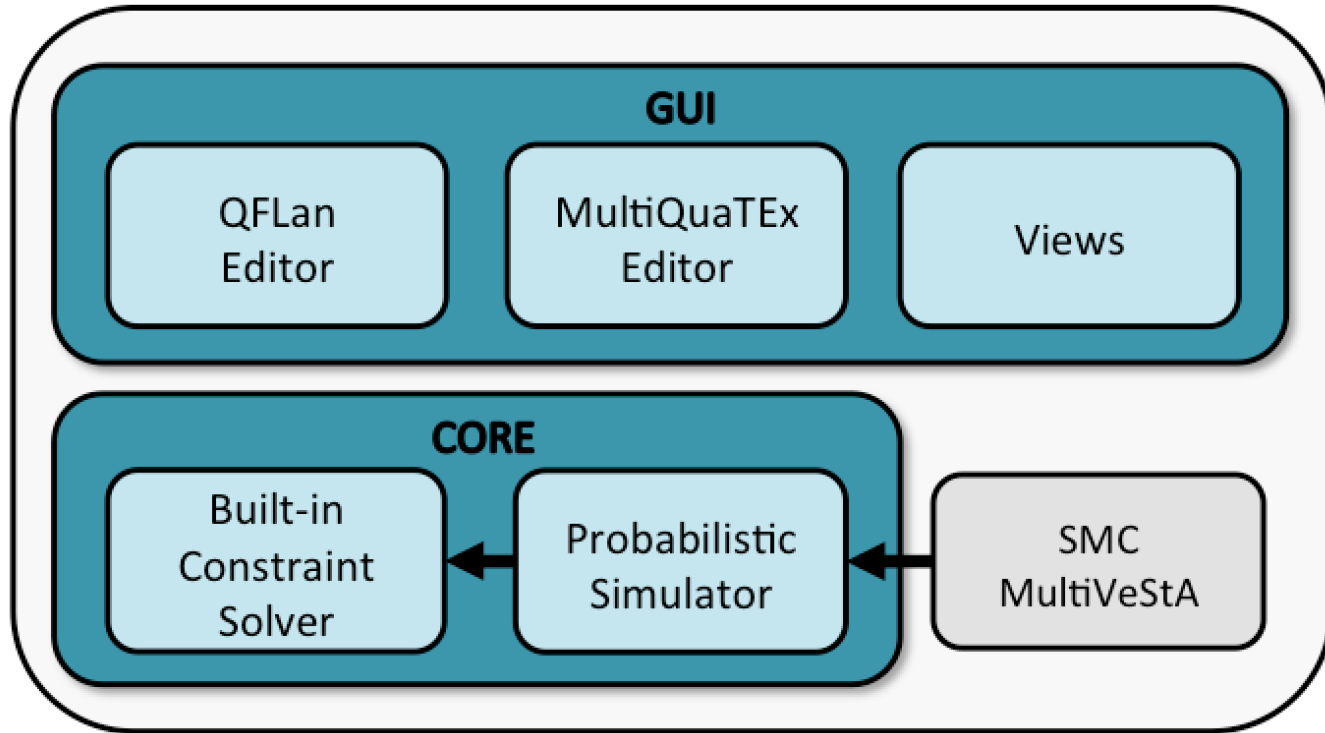
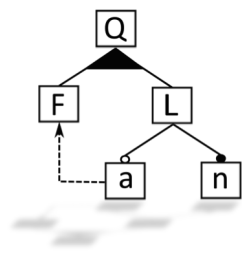


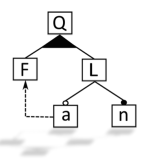
The screenshot displays the QFLan IDE interface with the following components:

- Project Explorer:** Shows a tree view of the project structure, including benchmarks, faze2018, MultiVeStA_OUTPUT, src-gen, VendingMachine.qflan, OtherQFLanProject, and QFLAN_Examples.
- QFLan Editor:** Contains the source code for the VendingMachine model, including variable declarations, abstract and concrete features, and a feature diagram.
- Outline View:** Provides a hierarchical view of the model's structure, including variables, abstract features, concrete features, feature relations, constraints, and actions.
- Console View:** Displays the output of the MultiVeStA analysis, showing simulation progress and results.
- Plot View:** Shows a line graph of means estimations over time (x-axis) for five different observations (obs1AtStep(x) to obs5AtStep(x)).



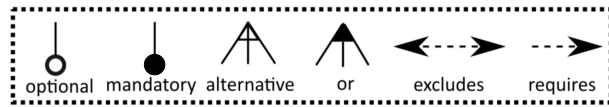
Presented in [FM'18][TSE'18]
Prototypes in [FMSPLE'15][SPLC'15][ISOLA'16]



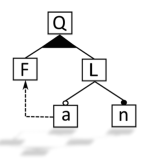


A simple vending machine product line

The feature model

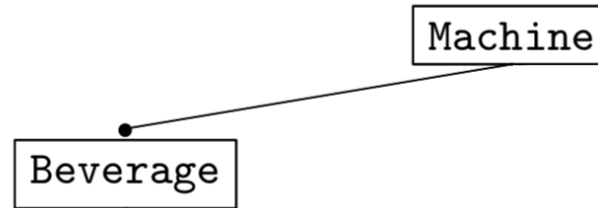
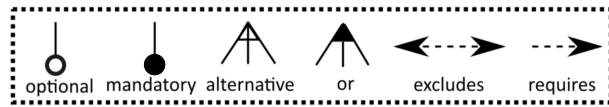


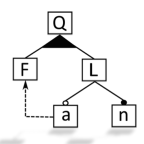
Machine



A simple vending machine product line

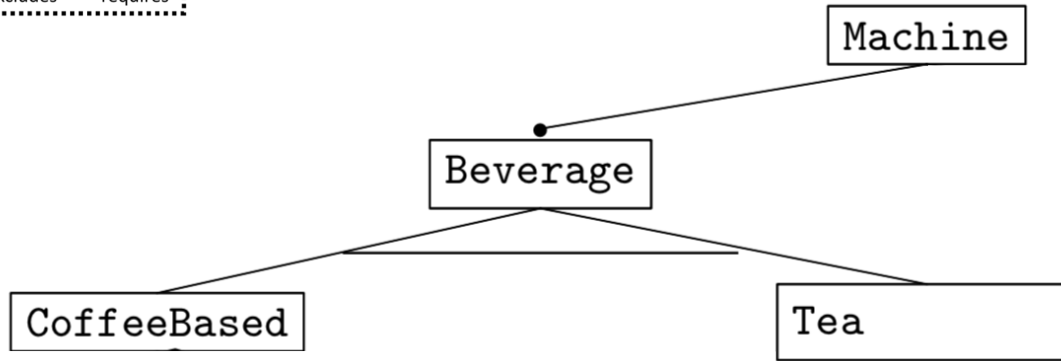
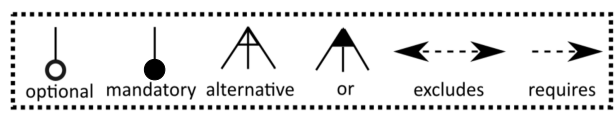
The feature model

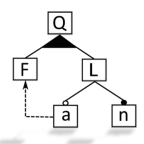




A simple vending machine product line

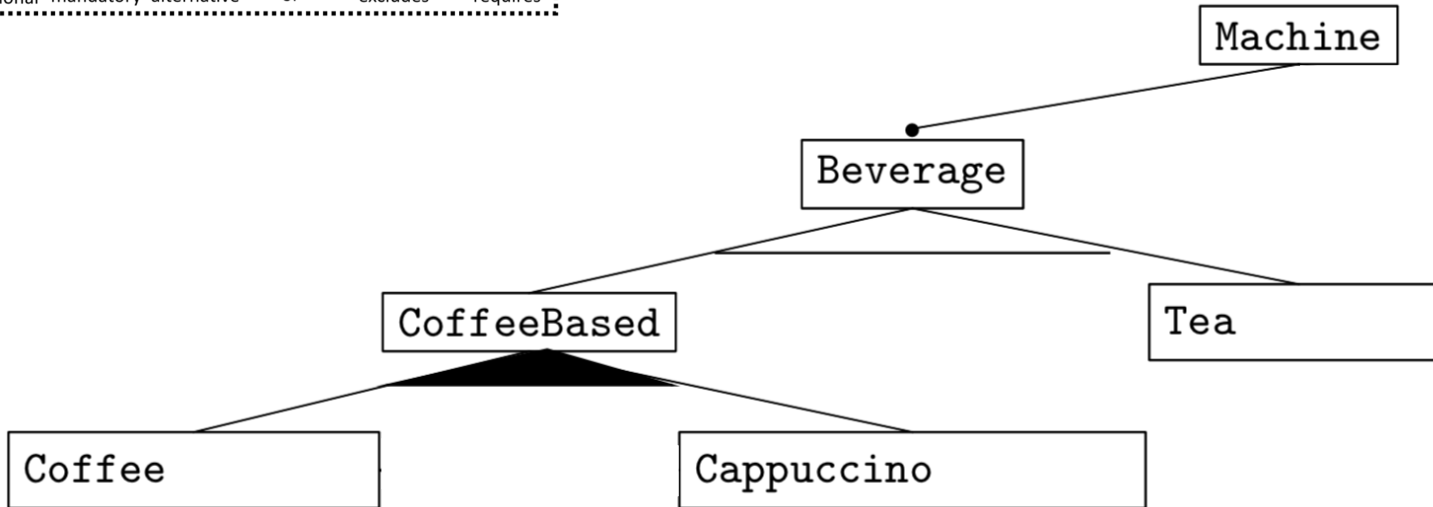
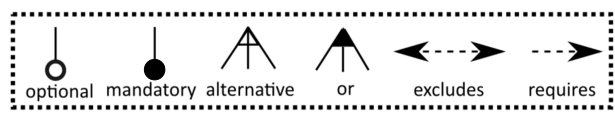
The feature model

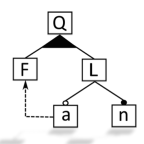




A simple vending machine product line

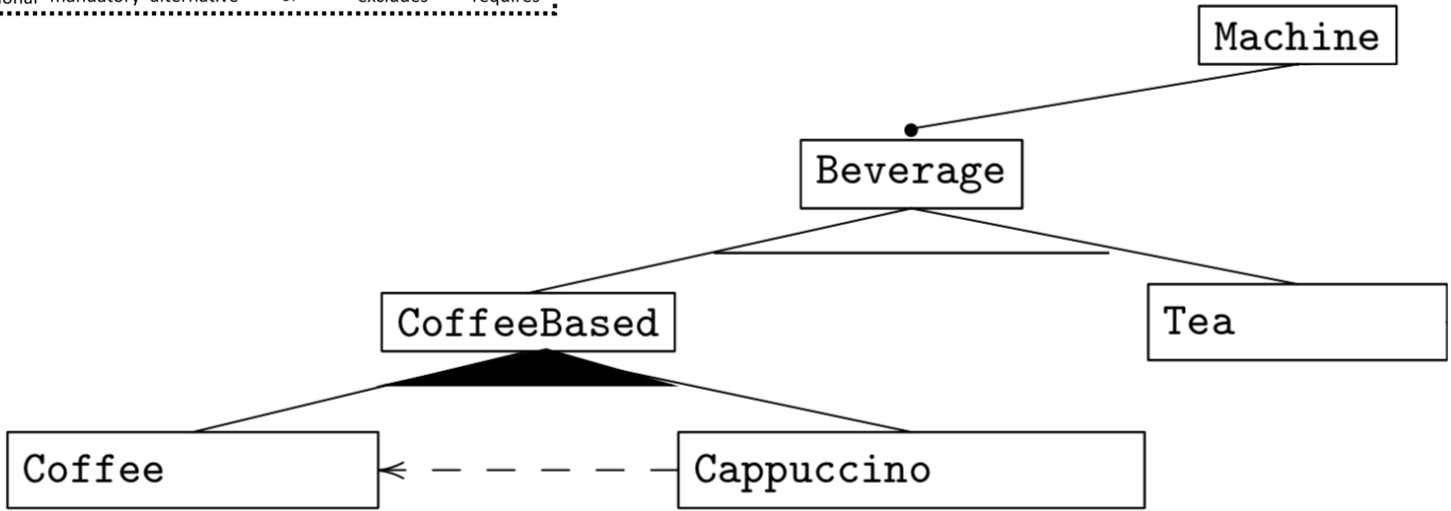
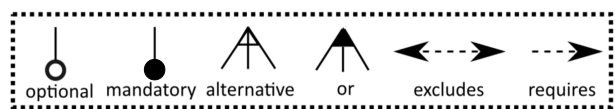
The feature model

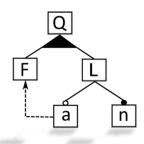




A simple vending machine product line

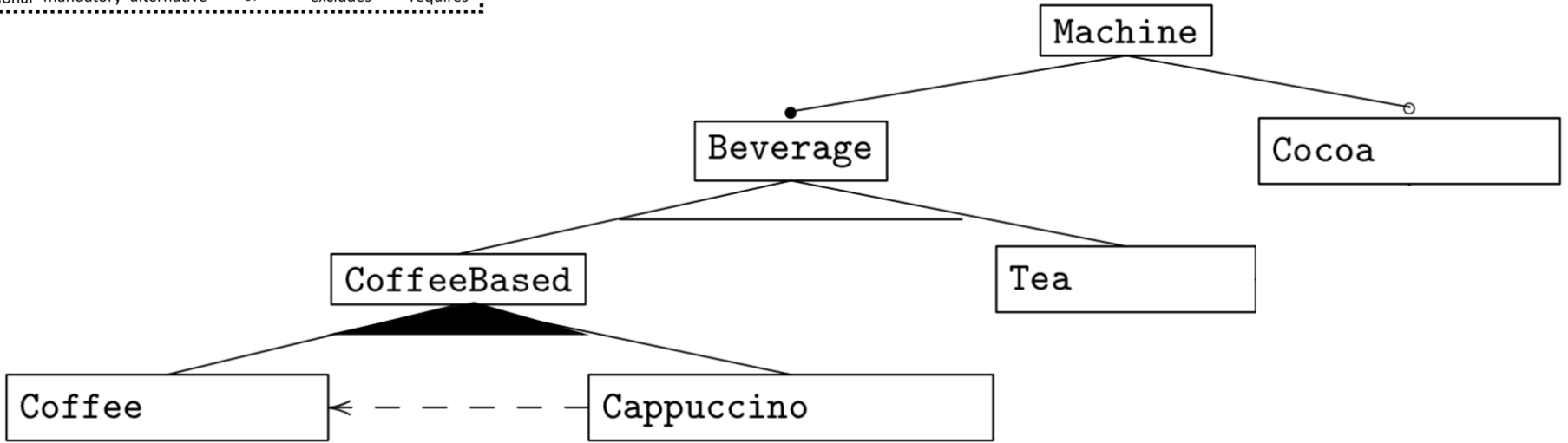
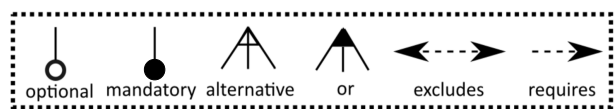
The feature model





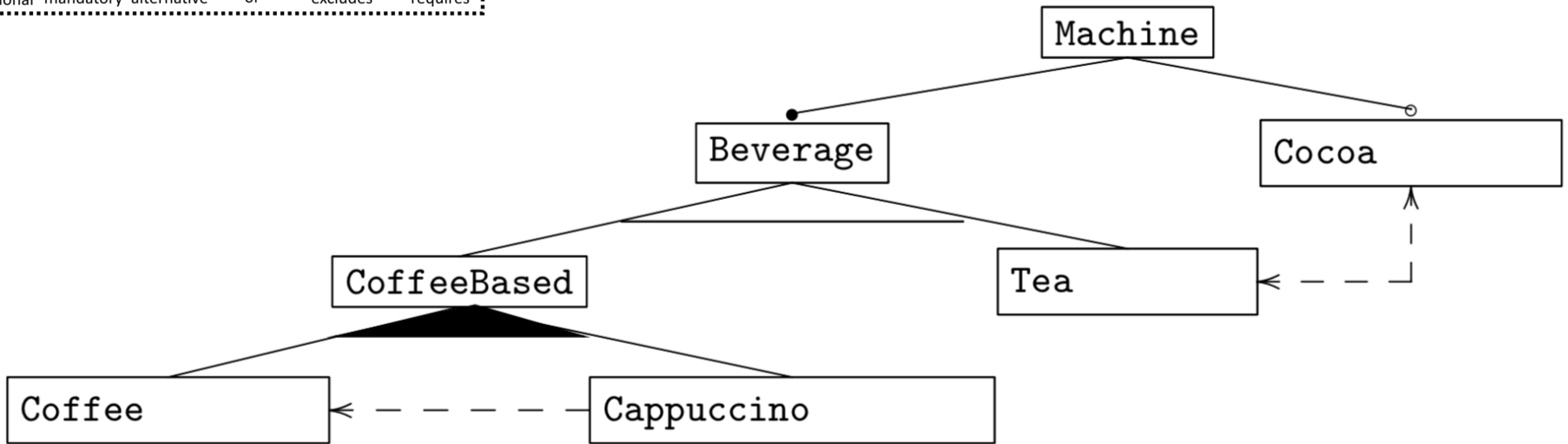
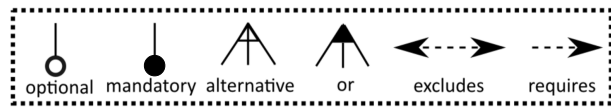
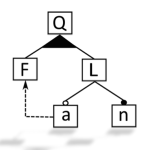
A simple vending machine product line

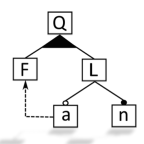
The feature model



A simple vending machine product line

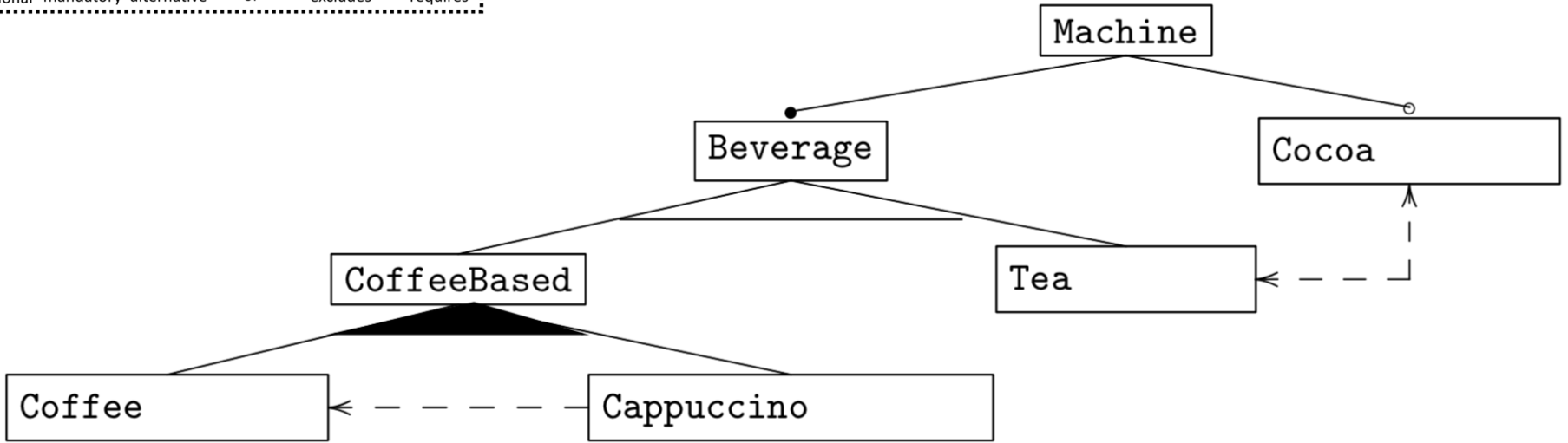
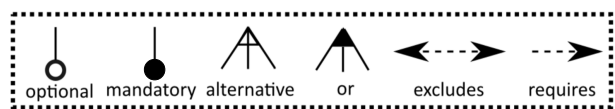
The feature model





A simple vending machine product line

The feature model: Abstract & Concrete Features



```

begin abstract features
Machine Beverage CoffeeBased
end abstract features

```

```

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

```

```

begin concrete features
Cocoa Tea Cappuccino Coffee
end concrete features

```

```

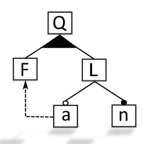
begin feature diagram
Machine -> {?Cocoa, Beverage}
Beverage -XOR-> {CoffeeBased,Tea}
CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram

```

```

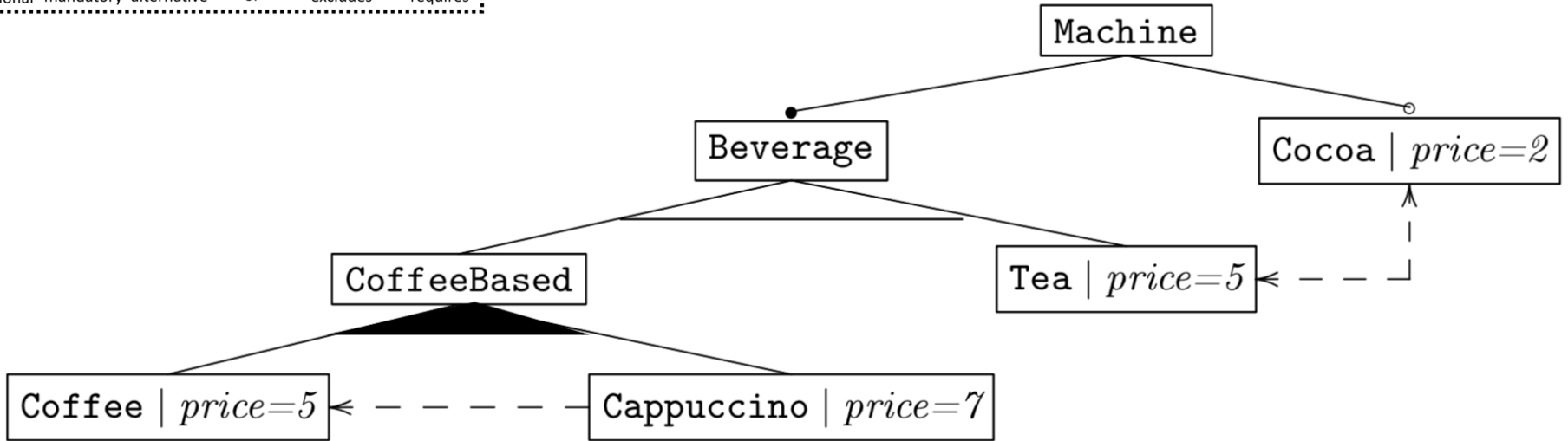
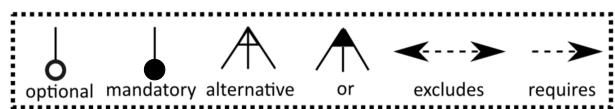
begin cross-tree constraints
Cappuccino requires Coffee
Tea excludes Cocoa
end cross-tree constraints

```



A simple vending machine product line

The feature model: Abstract & Concrete Features



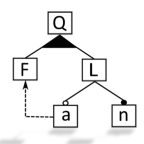
```
begin abstract features
Machine Beverage CoffeeBased
end abstract features
```

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

```
begin concrete features
Cocoa Tea Cappuccino Coffee
end concrete features
```

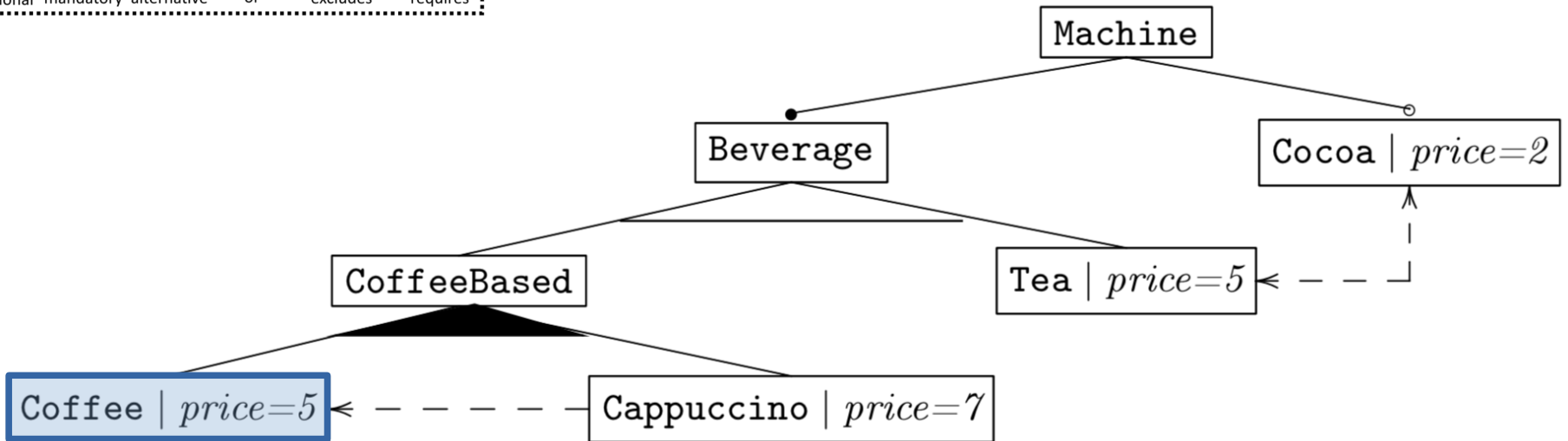
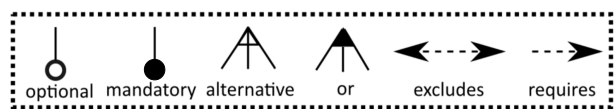
```
begin feature diagram
Machine -> {?Cocoa, Beverage}
Beverage -XOR-> {CoffeeBased,Tea}
CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram
```

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



A simple vending machine product line

The feature model: Abstract & Concrete Features



```

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

```

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

```

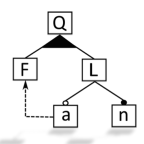
begin concrete features
Cocoa Tea Cappuccino Coffee
end concrete features
  
```

```

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

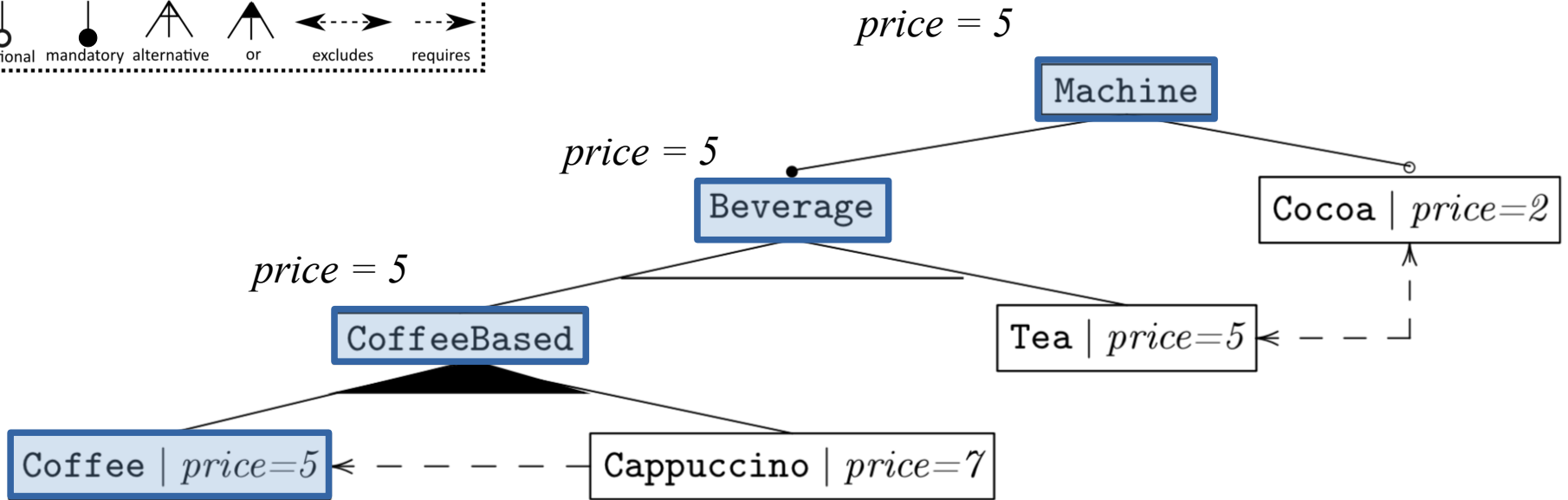
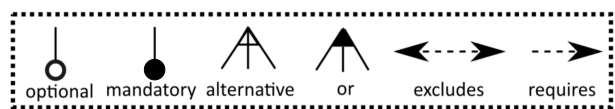
```

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



A simple vending machine product line

The feature model: Abstract & Concrete Features



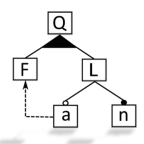
```
begin abstract features
Machine Beverage CoffeeBased
end abstract features
```

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

```
begin concrete features
Cocoa Tea Cappuccino Coffee
end concrete features
```

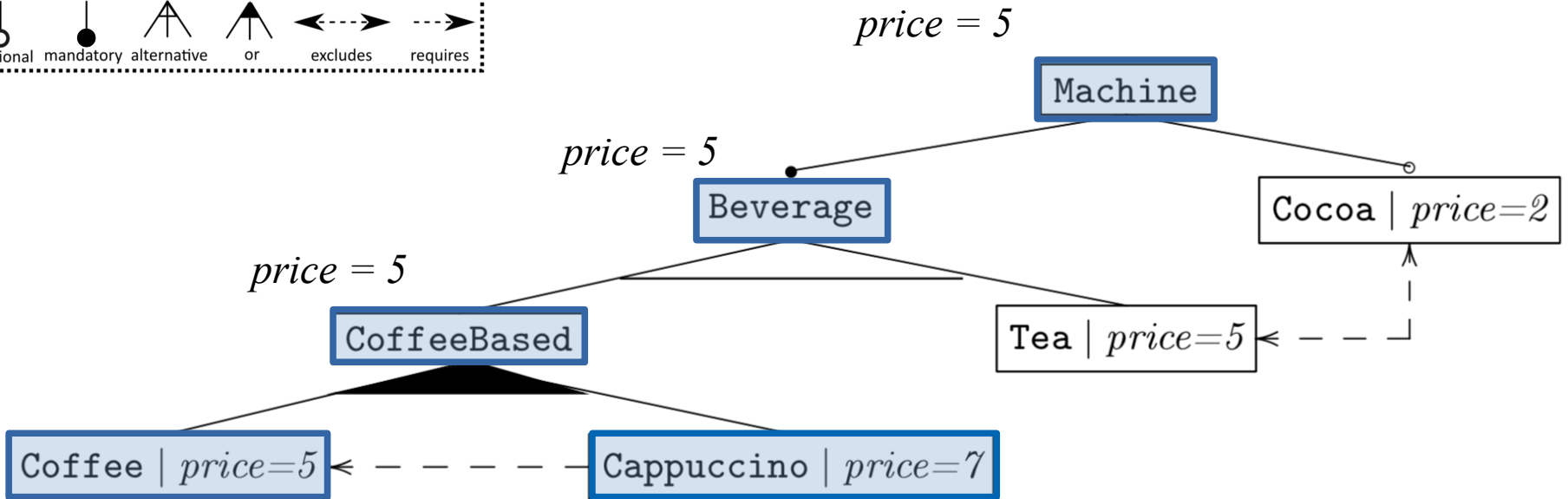
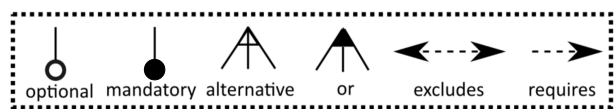
```
begin feature diagram
Machine -> {?Cocoa, Beverage}
Beverage -XOR-> {CoffeeBased,Tea}
CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram
```

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



A simple vending machine product line

The feature model: Abstract & Concrete Features



```

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

```

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

```

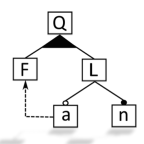
begin concrete features
Cocoa Tea Cappuccino Coffee
end concrete features
  
```

```

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

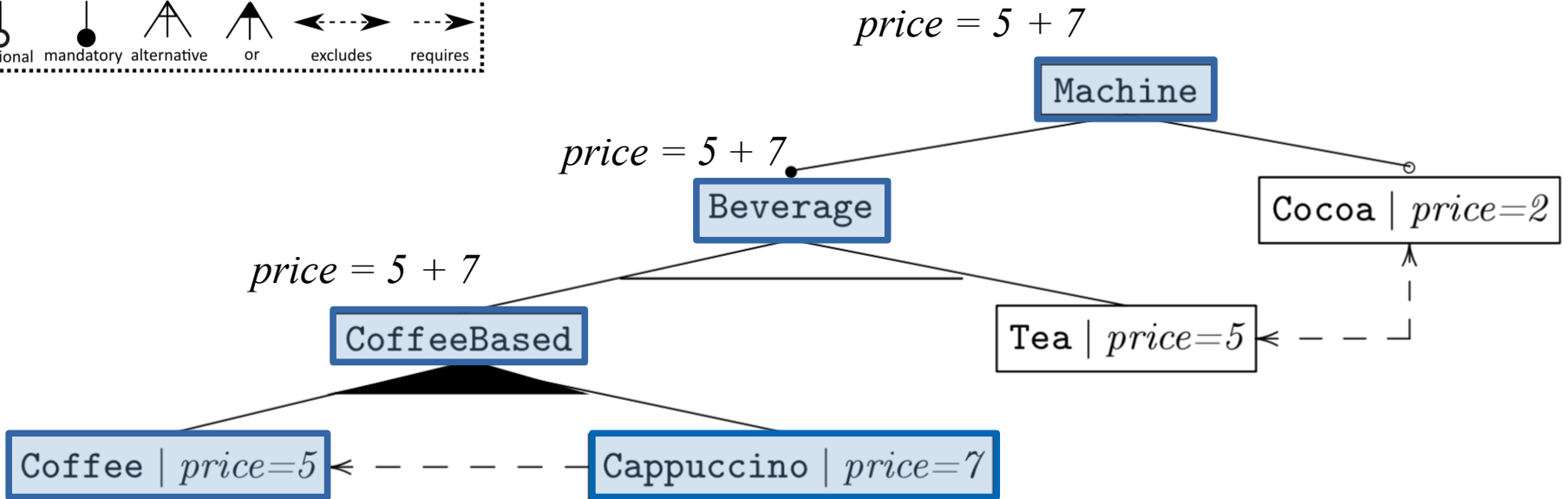
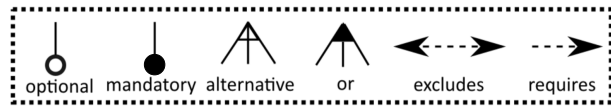
```

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

A simple vending machine product line

The feature model: Abstract & Concrete Features



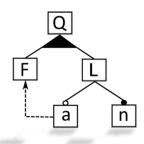
```
begin abstract features
Machine Beverage CoffeeBased
end abstract features
```

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

```
begin concrete features
Cocoa Tea Cappuccino Coffee
end concrete features
```

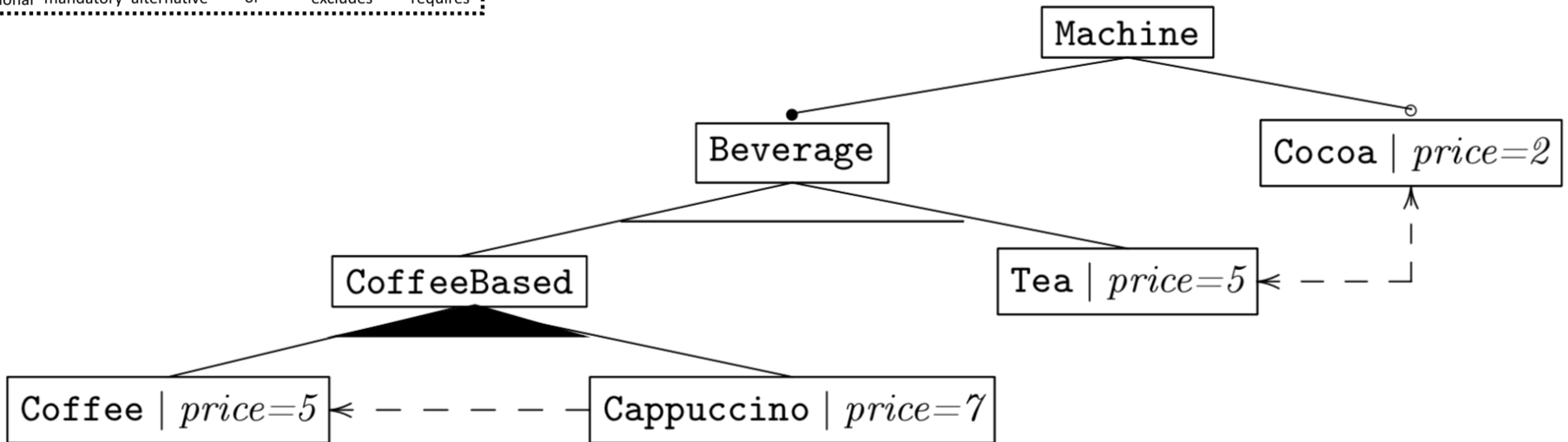
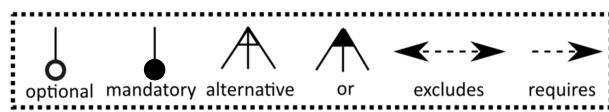
```
begin feature diagram
Machine -> {?Cocoa, Beverage}
Beverage -XOR-> {CoffeeBased,Tea}
CoffeeBased -OR->{Cappuccino,Coffee}
end feature diagram
```

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



A simple vending machine product line

The feature model: Quantitative constraints



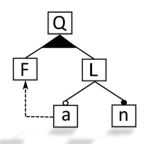
```
begin abstract features
Machine Beverage CoffeeBased
end abstract features
```

```
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
```

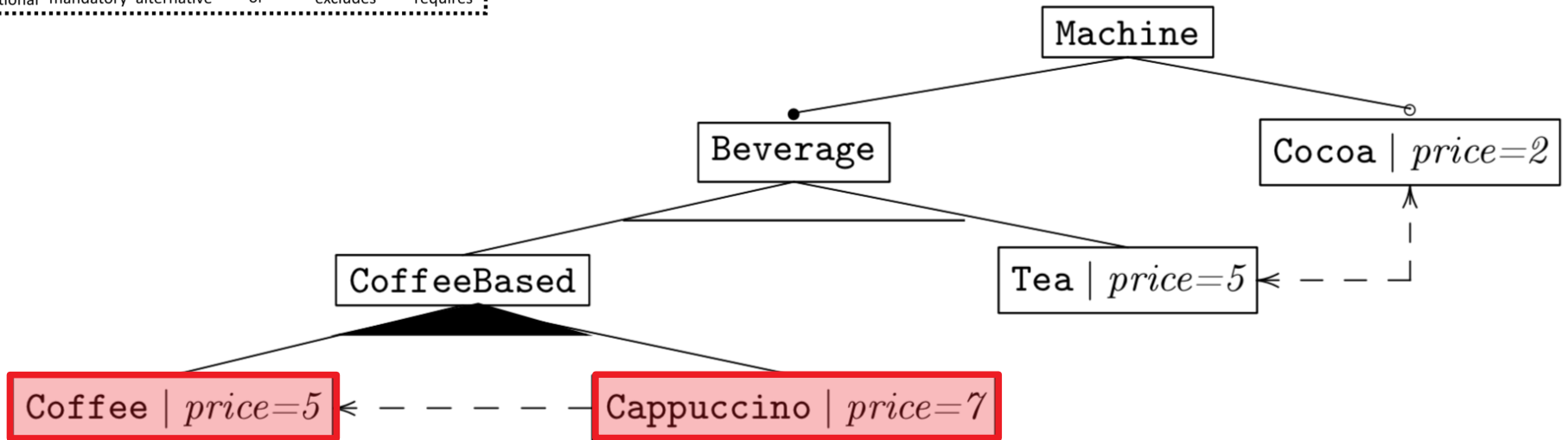
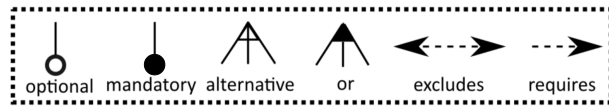
```
begin cross-tree constraints
Cappuccino requires Coffee
Tea excludes Cocoa
end cross-tree constraints
```

```
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



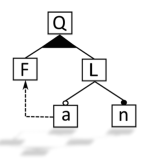
```
begin abstract features
Machine Beverage CoffeeBased
end abstract features
```

```
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
```

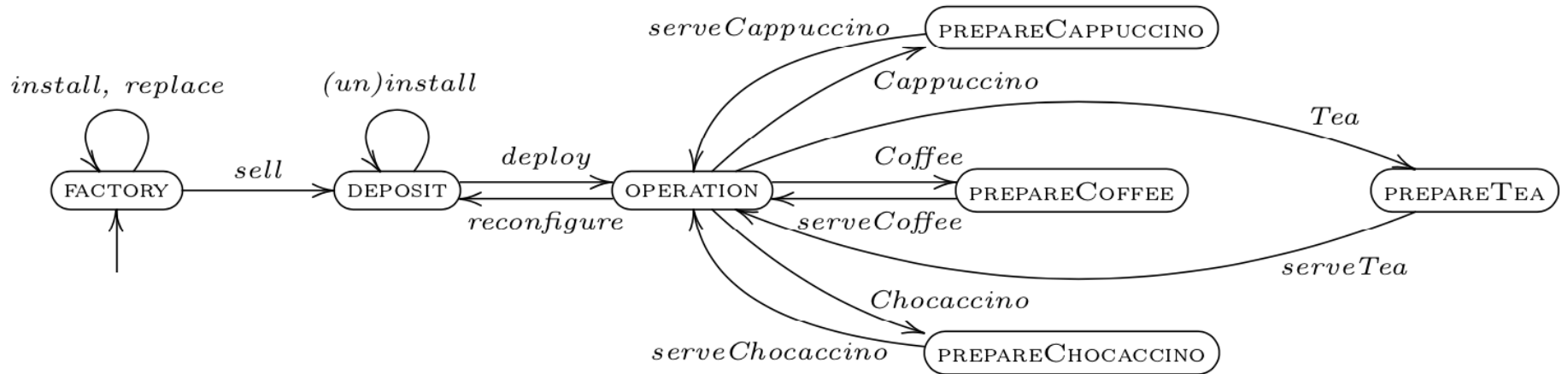
```
begin cross-tree constraints
Cappuccino requires Coffee
Tea excludes Cocoa
end cross-tree constraints
```

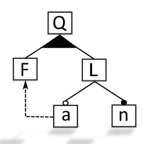
```
begin feature predicates
price= { Cappuccino = 7, Coffee = 5,
Cocoa = 2, Tea = 5 }
end feature predicates
begin quantitative constraints
{ price(Machine) <= 10 }
end quantitative constraints
```



A simple vending machine product line

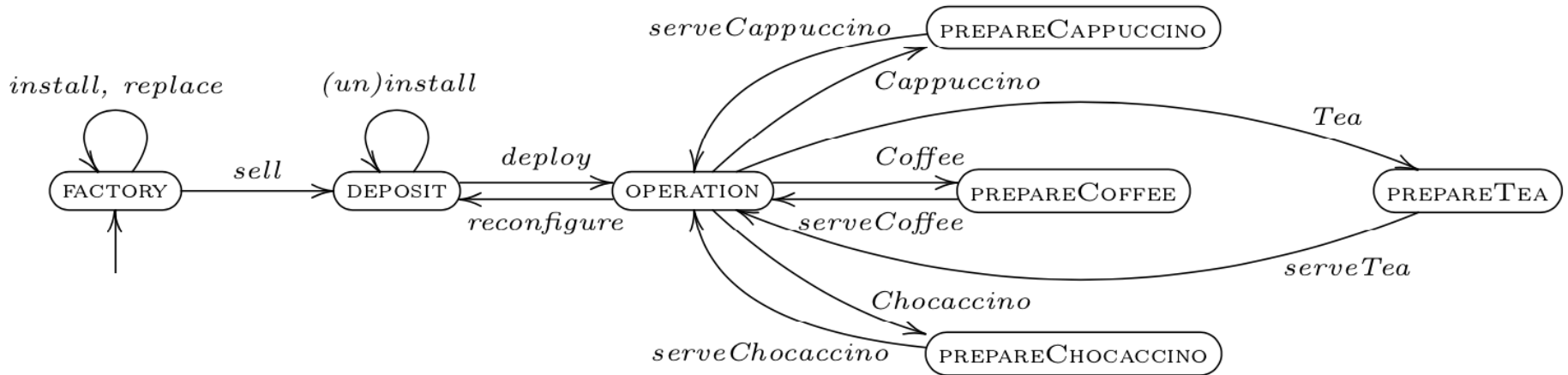
Behaviour: actions and action constraints





A simple vending machine product line

Behaviour: actions and action constraints

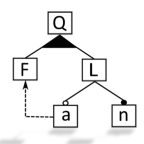


```

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

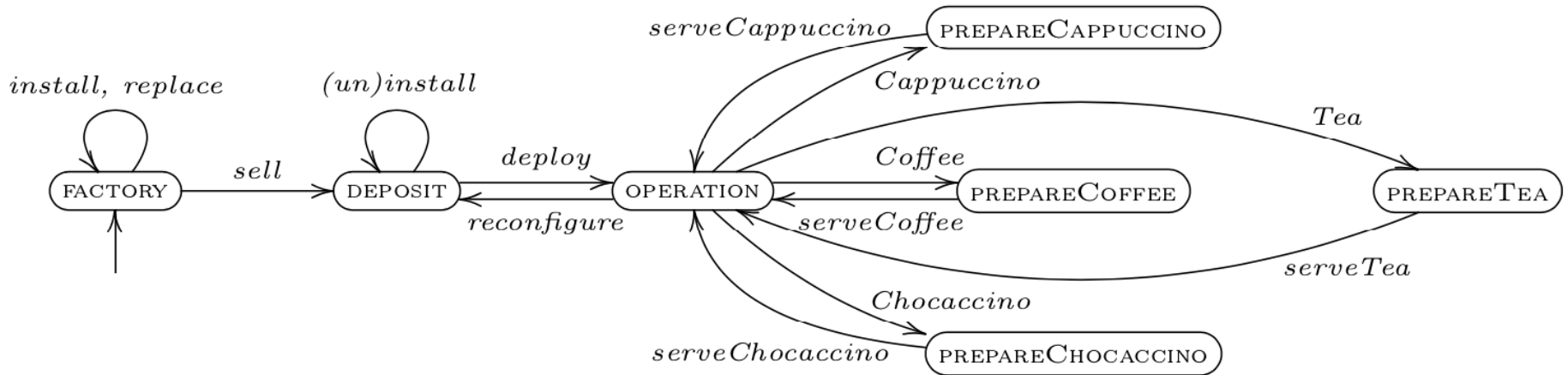
```

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



A simple vending machine product line

Behaviour: transitions

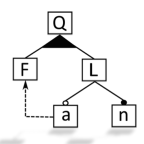


begin processes diagram
begin process dynamics

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

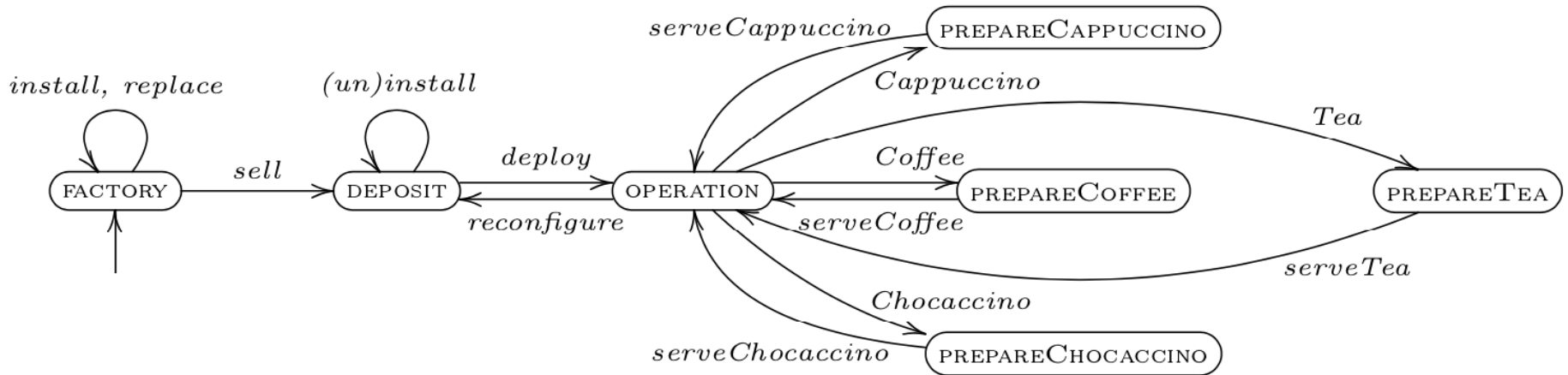
begin actions
sell deploy reconfigure
chocaccino
serveCoffee serveCappuccino
serveChocaccino serveTea
end actions

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



A simple vending machine product line

Behaviour: transitions



begin 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 ,
```

begin actions

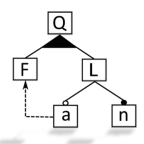
```
sell deploy reconfigure
chocaccino
serveCoffee serveCappuccino
serveChocaccino serveTea
```

end actions

begin action constraints

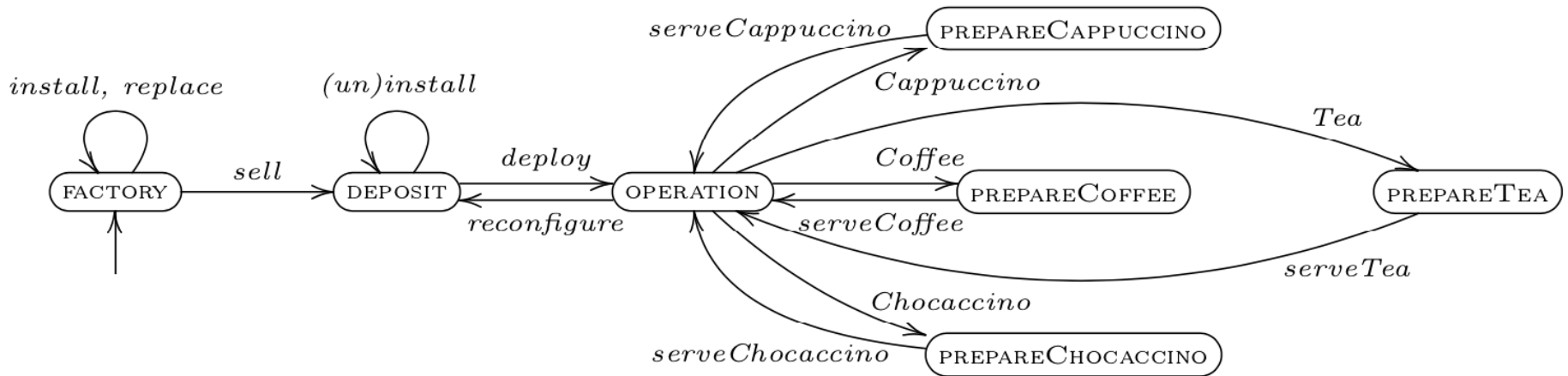
```
do(chocaccino) -> (has(Cappuccino)
and has(Cocoa) )
```

end action constraints



A simple vending machine product line

Behaviour: transitions



begin 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

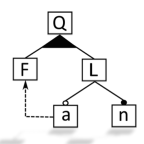
states = factory , deposit , operating , prepareCoffee ,
prepareCappuccino , prepareTea , prepareChocaccino

transitions =

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

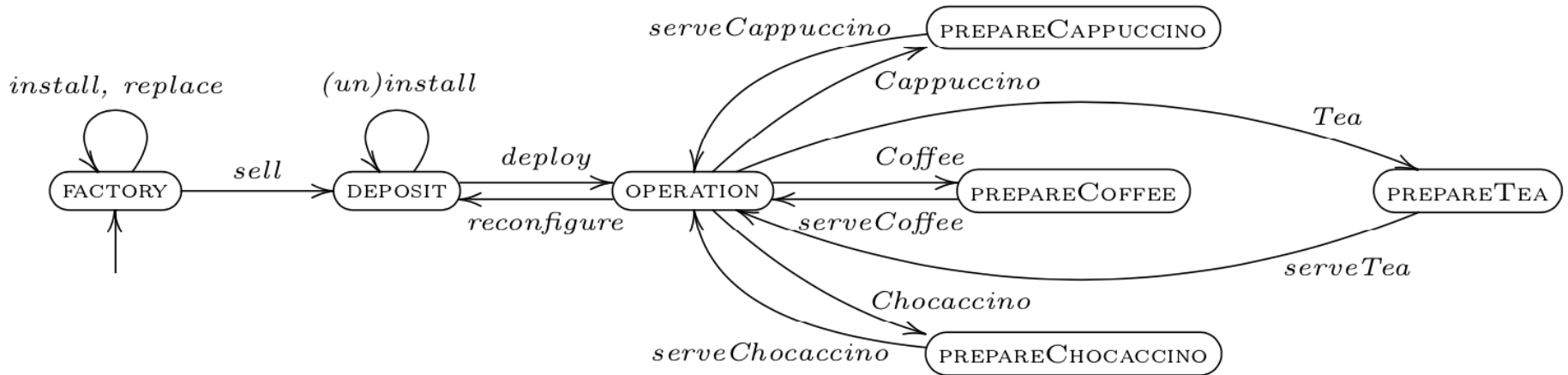
//Deposit

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

```

```

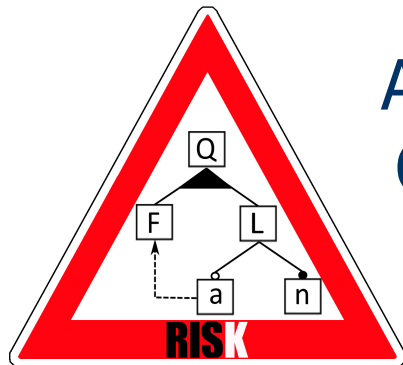
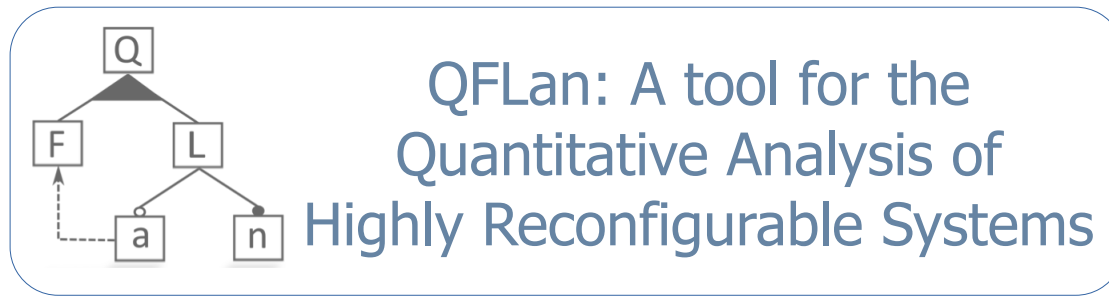
begin processes diagram
begin process dynamics
states = factory , deposit , operating , prepareCoffee ,
prepareCappuccino, prepareTea , prepareChocaccino
transitions =
//Factory
factory -(replace(Coffee,Tea),20)->factory,
factory -(install(Cocoa),10)->factory,
factory -(install(Cappuccino),10)->factory,
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

```



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

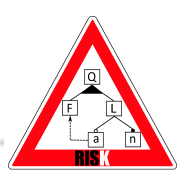
Andrea Vandin

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

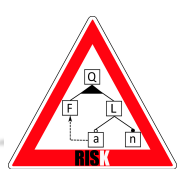


A Bank Robbery Scenario in RisQFLan

A screenshot of RisQFLan

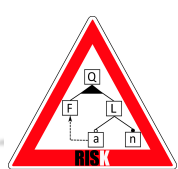
The screenshot displays the RisQFLan software interface with several components:

- Project Explorer:** Shows the project structure with folders for modelsMainPaper, modelsPaper, modelsToolPaper, and test.
- RisQFLan Editor:** Contains the model definition for RobBank, including variables, attack nodes, defense nodes, countermeasure nodes, and an attack diagram. The attack diagram shows a sequence of actions: RobBank leads to OpenVault and BlowUp, which then leads to LearnCombo and GetToVault, and finally to FindCode1, FindCode2, and FindCode3. A LockDown node is also shown as a defense node.
- Plot View:** Displays a line graph titled "Mean estimations" showing the results of an iViVeSTA analysis. The x-axis represents the number of iterations (0 to 100), and the y-axis represents the mean estimation (0 to 0.93). Multiple lines represent different nodes and their convergence over time.
- Tree View:** Shows a decision tree for the RobBank scenario. The root node is RobBank, which branches into OpenVault and BlowUp. OpenVault branches into LearnCombo and GetToVault. LearnCombo branches into FindCode1, FindCode2, and FindCode3. BlowUp branches into LockDown and LaserCutter. The LockDown node is a diamond shape, indicating a defense node. The LaserCutter node is a circle shape, indicating an attack node. The Memo node is a green rectangle, indicating a memo node.
- Outline View:** Shows the structure of the model, including variables, attack nodes, defense nodes, countermeasure nodes, and the attack diagram.
- Console View:** Shows the output of the simulation, including the name of the model and the date and time of the simulation.



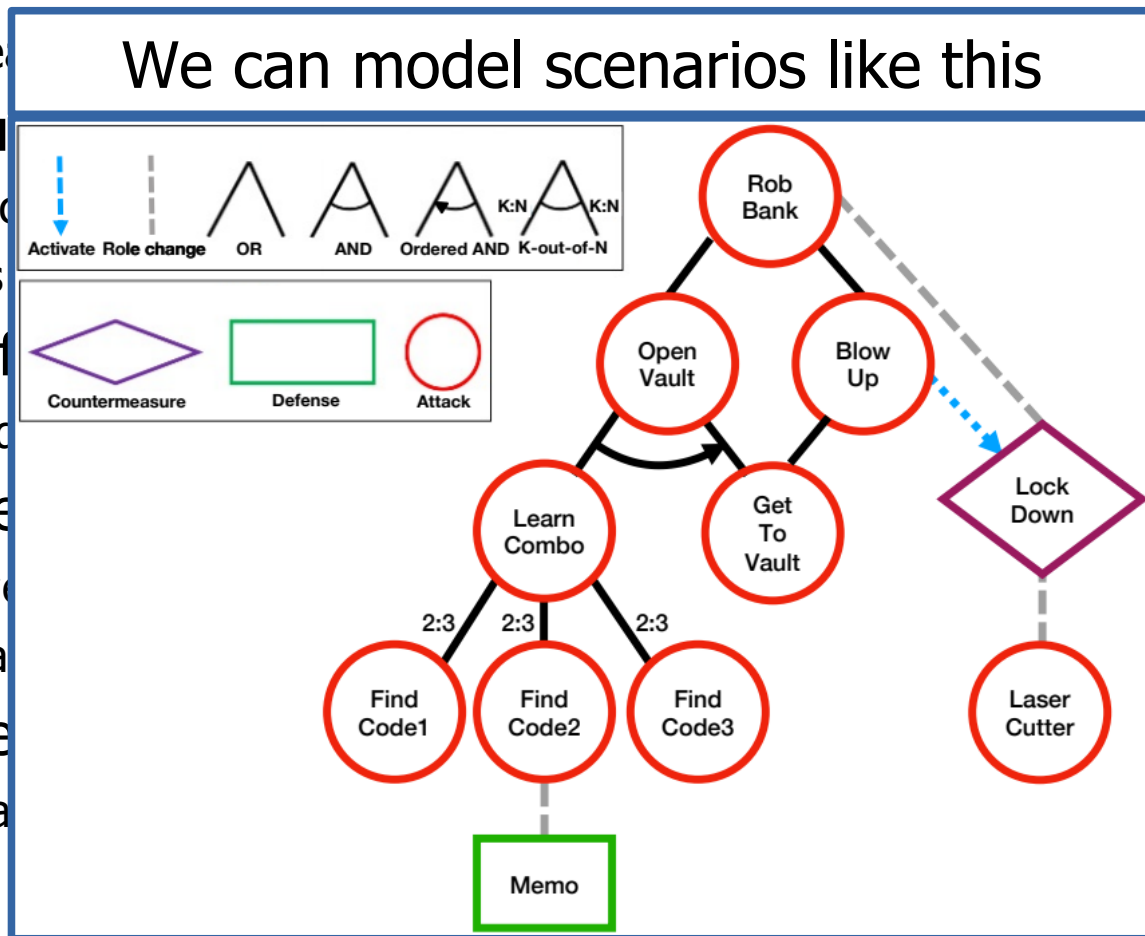
QFLan limitations for Risk Modeling/Analysis

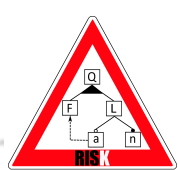
- Not entirely direct encoding of the scenario
 - The extra root node, the extra states to model failures, etc
- We need different types of nodes
 - Attack, defense, countermeasure
- We need richer constructs for building the tree diagram
 - QFLan has: or, requires, excludes
 - Missing *common* constructs: 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

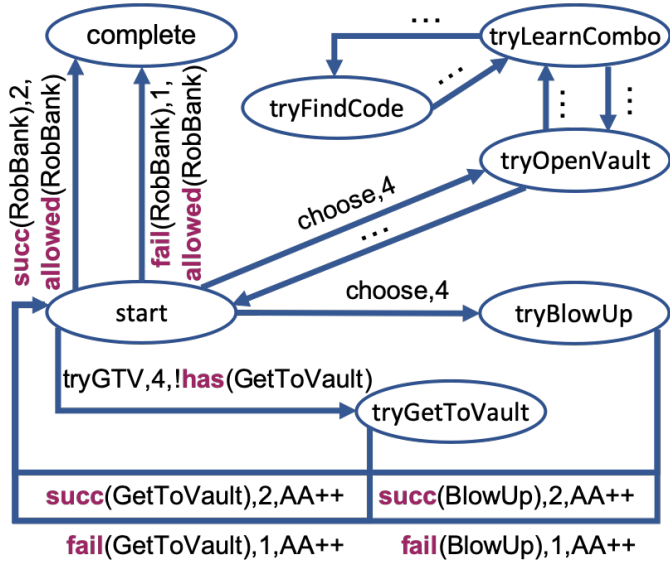
- Not entirely direct encoding of the scenario
 - The extra root node, the extra states to model failures, etc
- We need different types of nodes
 - Attack, defense, countermeasures
- We need richer constructs
 - QFLan has: or, requires, excludes
 - Missing *common* constructs
- Attack attempts might fail
 - The 'install' of an attack node
- There is no 'absolute security'
 - *Qualitative* constraints like 'very likely'
 - Often, failure probabilities are not available
- Exact analysis might be difficult
 - Complement MultiVeStA State





A Bank Robbery Scenario in RisQFLan

Analysis: SMC with MultiVeStA

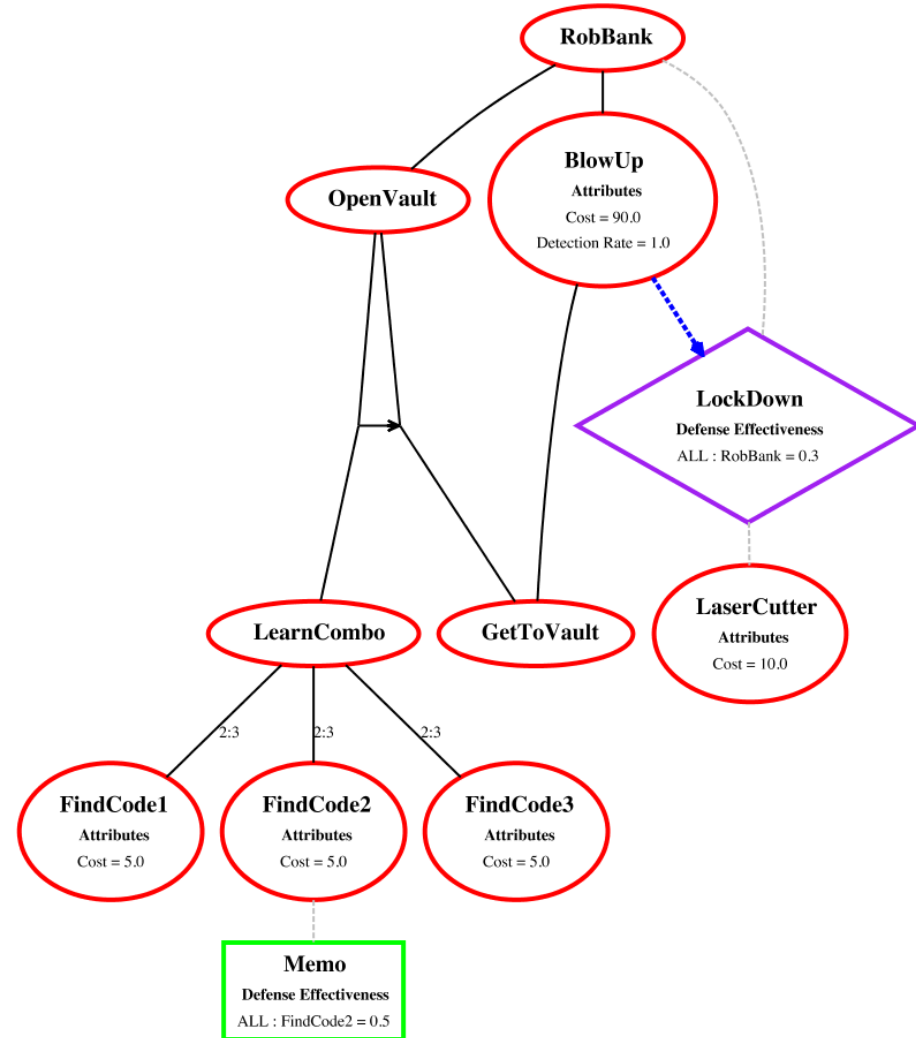
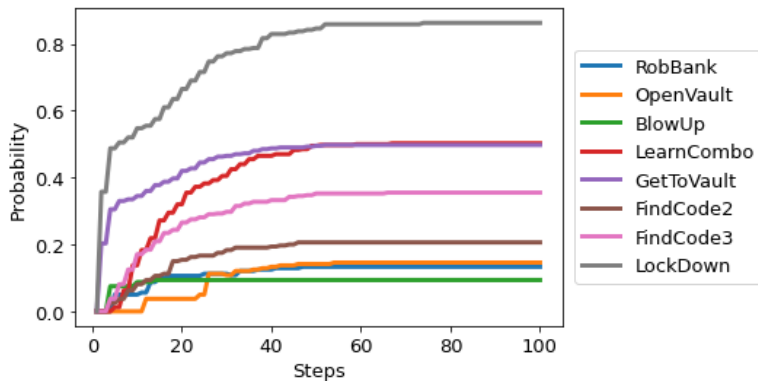


Attacker Behaviour

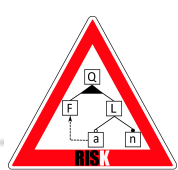
```

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

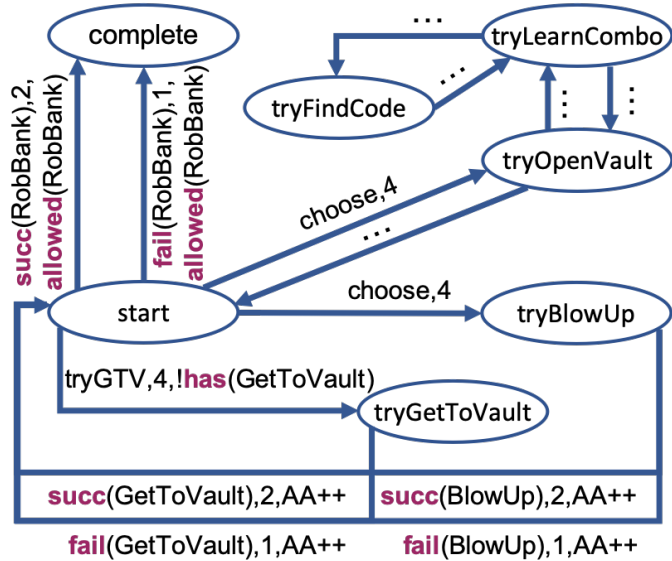


Attack-defense tree



A Bank Robbery Scenario in RisQFLan

Analysis: PMC with PRISM/STORM

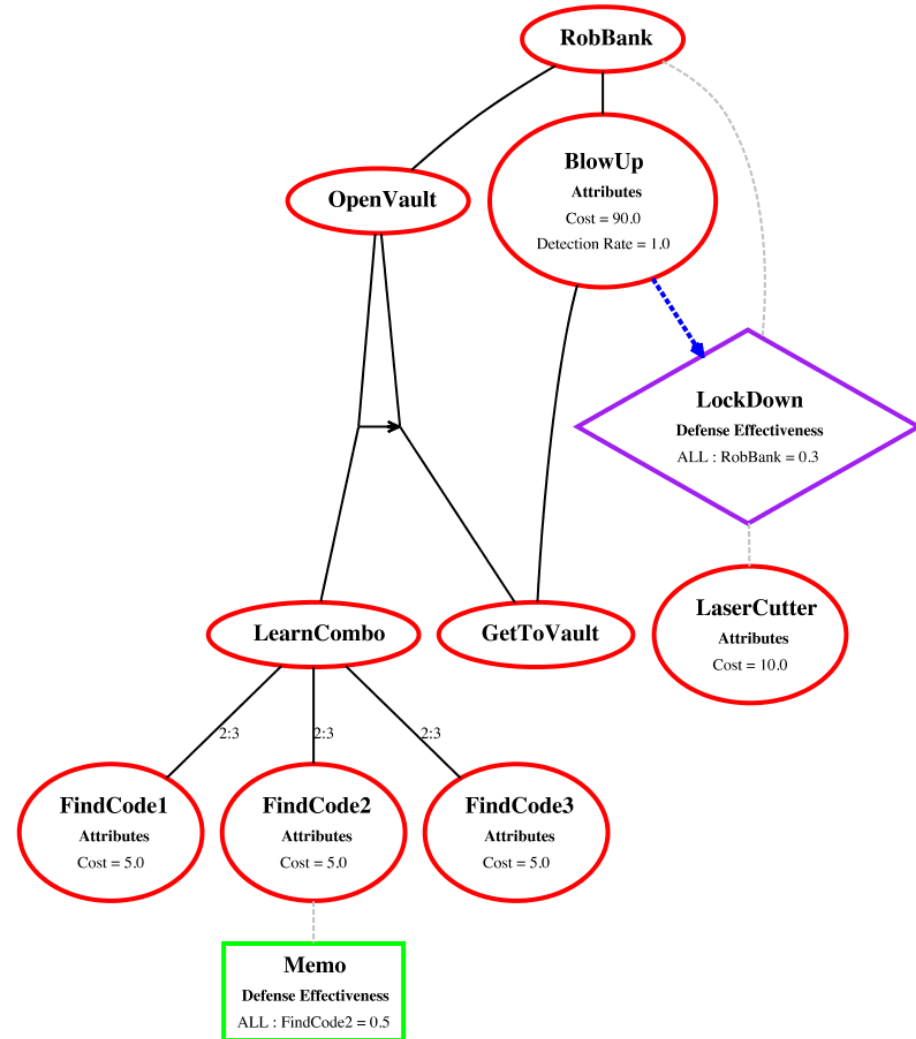
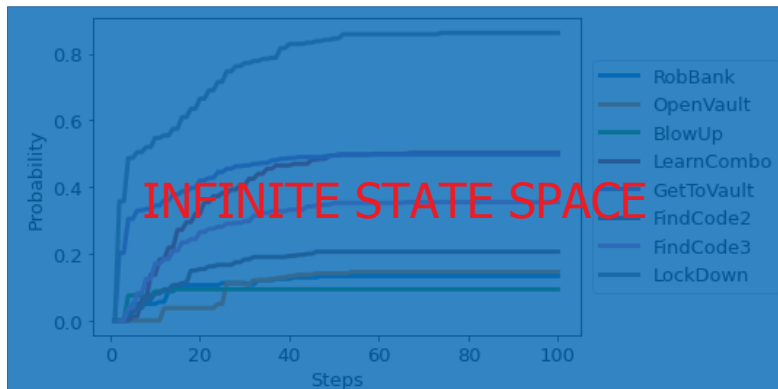


Attacker Behaviour

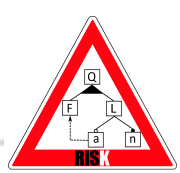
```

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

Exact
PMC Analysis

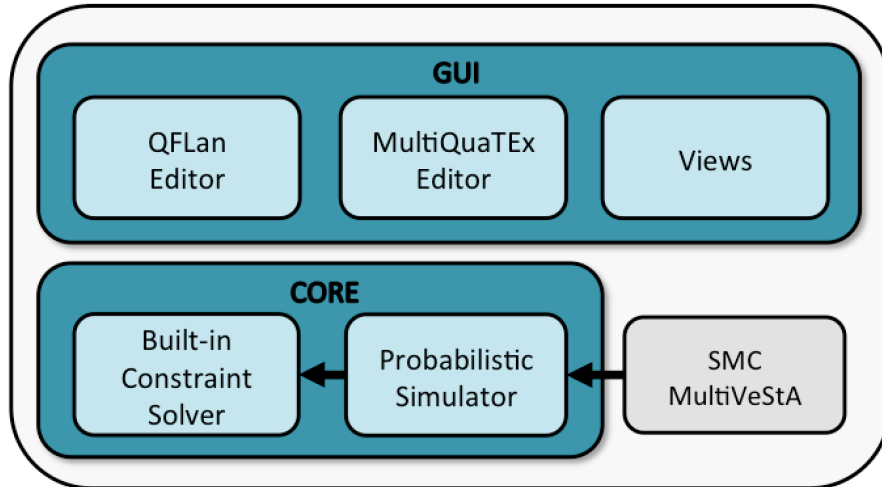


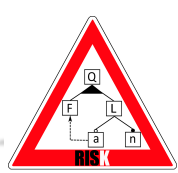
Attack-defense tree



From QFLan to RisQFLan Generalizing the QFLan approach

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

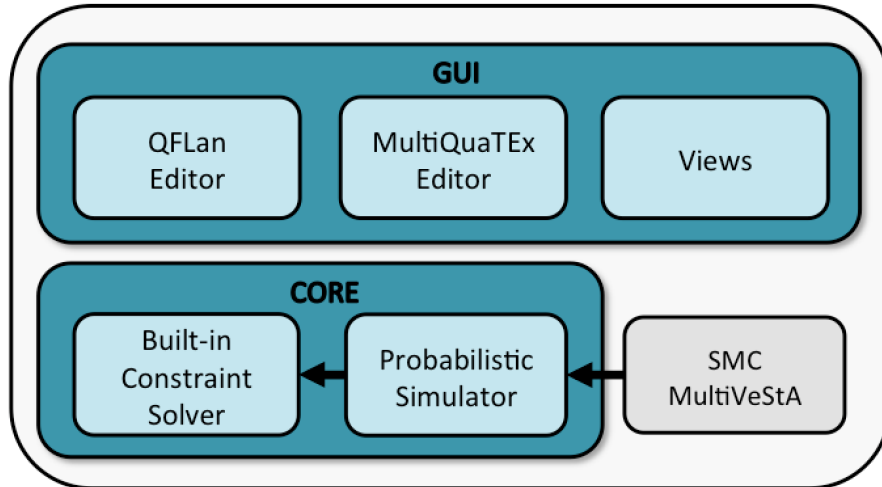




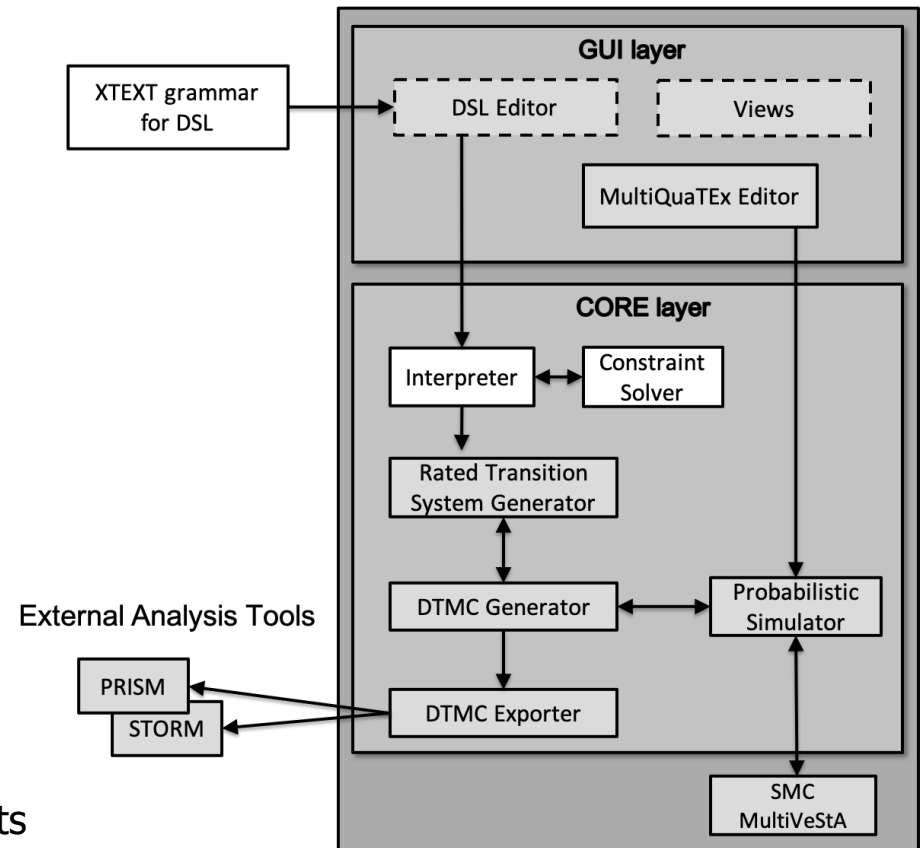
From QFLan to RisQFLan

Generalizing the QFLan approach

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



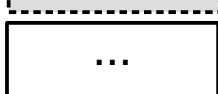
Generalized QFLan Architecture [COSE'21]



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

