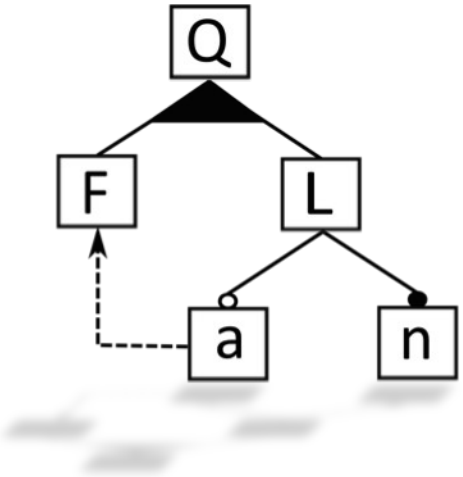



# A framework for quantitative modeling and analysis of highly reconfigurable systems

Maurice H. ter Beek
ISTI-CNR, Pisa, Italy

Axel Legay
UC Louvain, Belgium

Alberto Lluch Lafuente
DTU, Denmark

Andrea Vandin
DTU, Denmark
Sant'Anna School for Adv Studies Pisa, Italy

15th International Conference on integrated Formal Methods
5 December 2019, Bergen, Norway

# References

**[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.

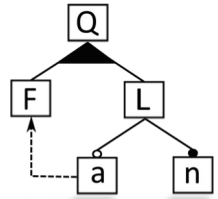
**[FM18]** Andrea Vandin, Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems.

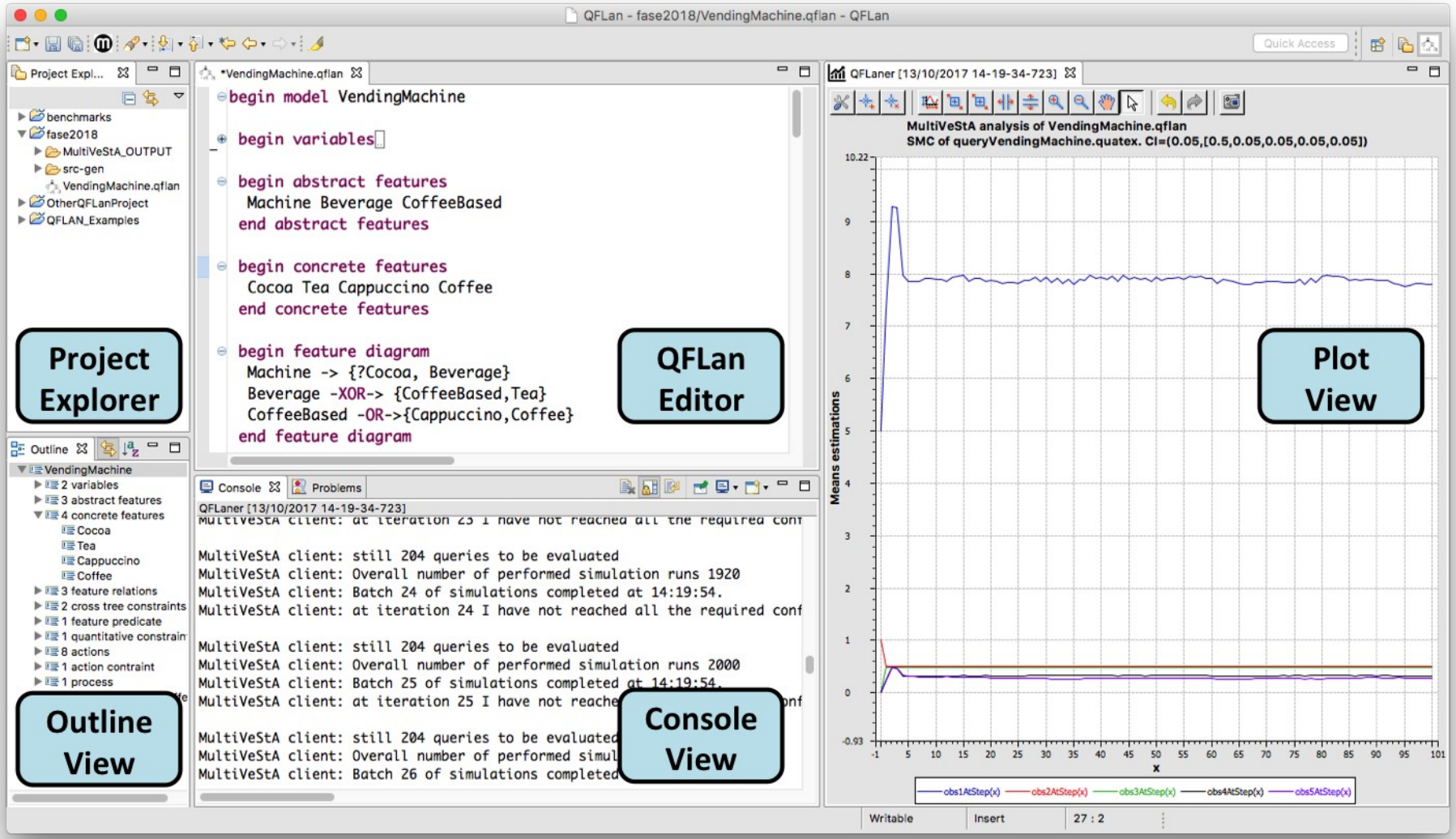
**[ISOLA16]** Maurice ter Beek, Axel Legay, Alberto Lluch Lafuente, Andrea Vandin, Statistical Model Checking for Product Lines.

**[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]**
**Invited survey in [ISOLA'16]**
**Prototypes in [FMSPLE'15][SPLC'15]**

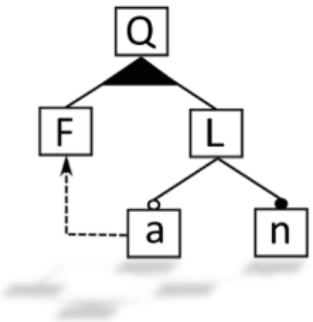



https://github.com/qflanTeam/QFLan/

# QFLan

*A framework for quantitative modeling and analysis of highly (re)configurable systems*




## Summary

*QFLan* is a software tool for the modeling and analysis of highly reconfigurable systems, including software product lines.

The tool offers an easy-to-use, rule-based probabilistic language to specify models with probabilistic behaviour. Quantitative constraints can be used to restrict the class of admissible configurations (or products), like (using a family of reconfigurable vending machines from here):

- machines can have a certain maximum cost,
- machines serving coffee-based beverages cannot sell tea,
- in order to serve cappuccino it is necessary to have the feature of serving also coffee,

Also it is possible to express conditions like:

- machines serving cappuccino provided with a coca dispenser can serve chocaccino.

QFLan has been combined with the distributed statistical model checker MultiVeStA to perform



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

# Outline

## Feature Model

- Abstract and Concrete Features
- Cross-tree Constraints
- Quantitative Constraints

## Behaviour

- Actions and Action Constraints
- Transitions
- Initial Configuration

## MultiVeStA Analysis

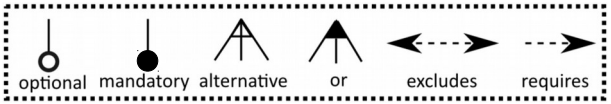
- Analysis when a condition holds
- Analysis at varying of time

## An Application to a Simple Security Scenario

- Schneier's SafeLock Attack Tree

# A simple vending machine product line

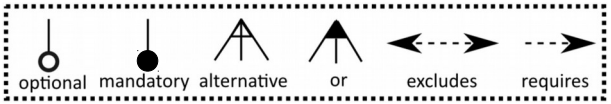
## The feature model

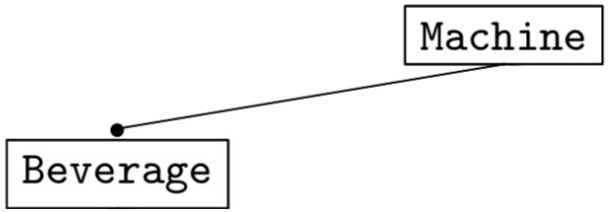



Machine

# A simple vending machine product line

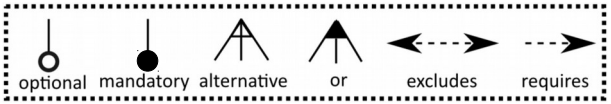
## The feature model

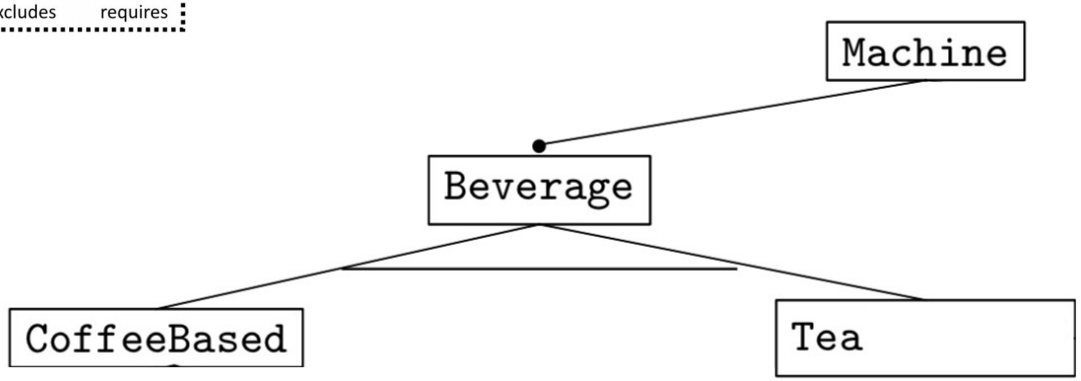


# A simple vending machine product line

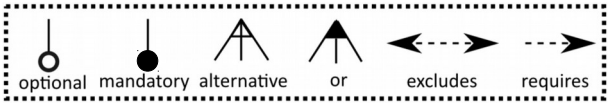
## The feature model

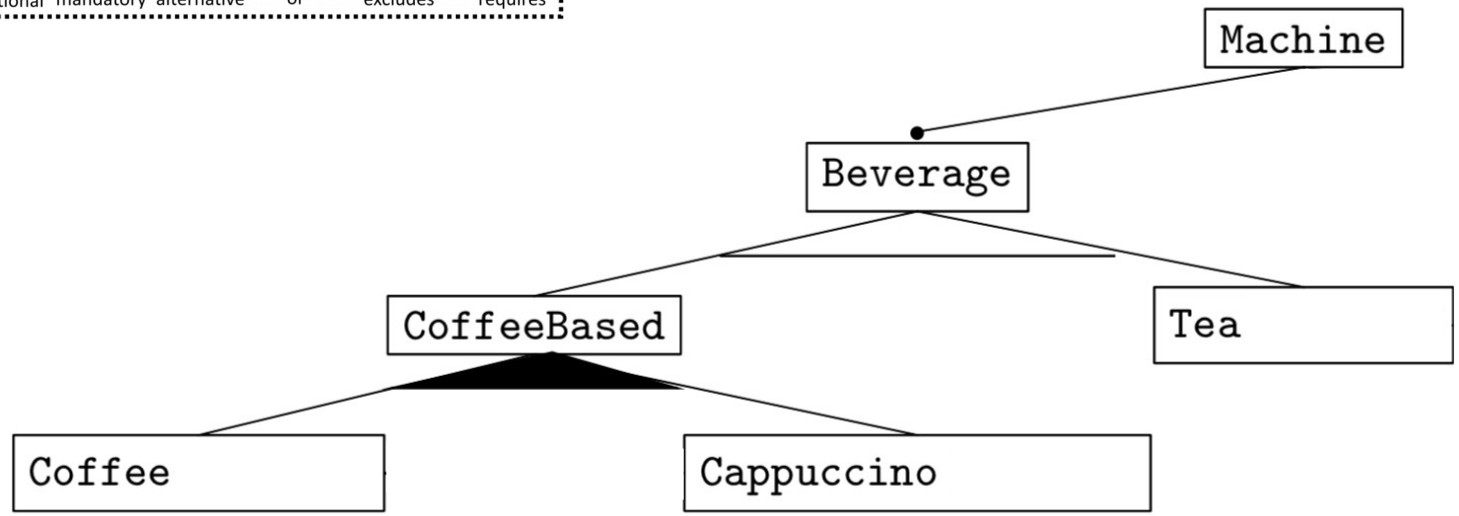


# A simple vending machine product line

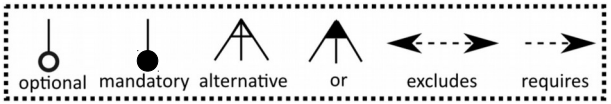
## The feature model

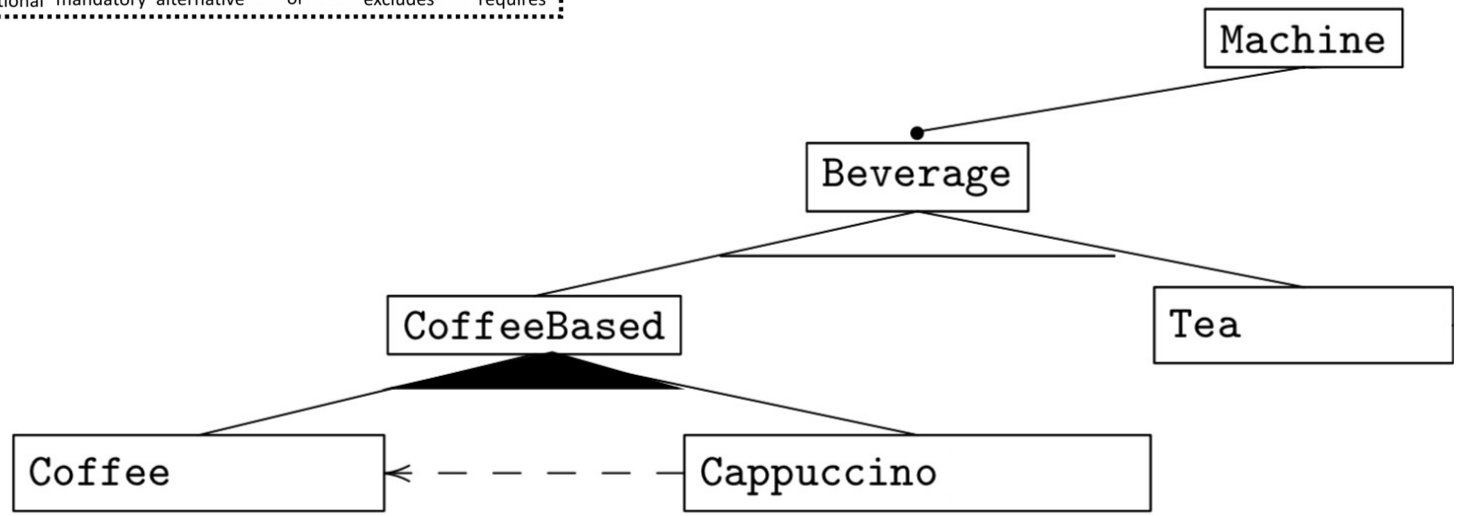


# A simple vending machine product line

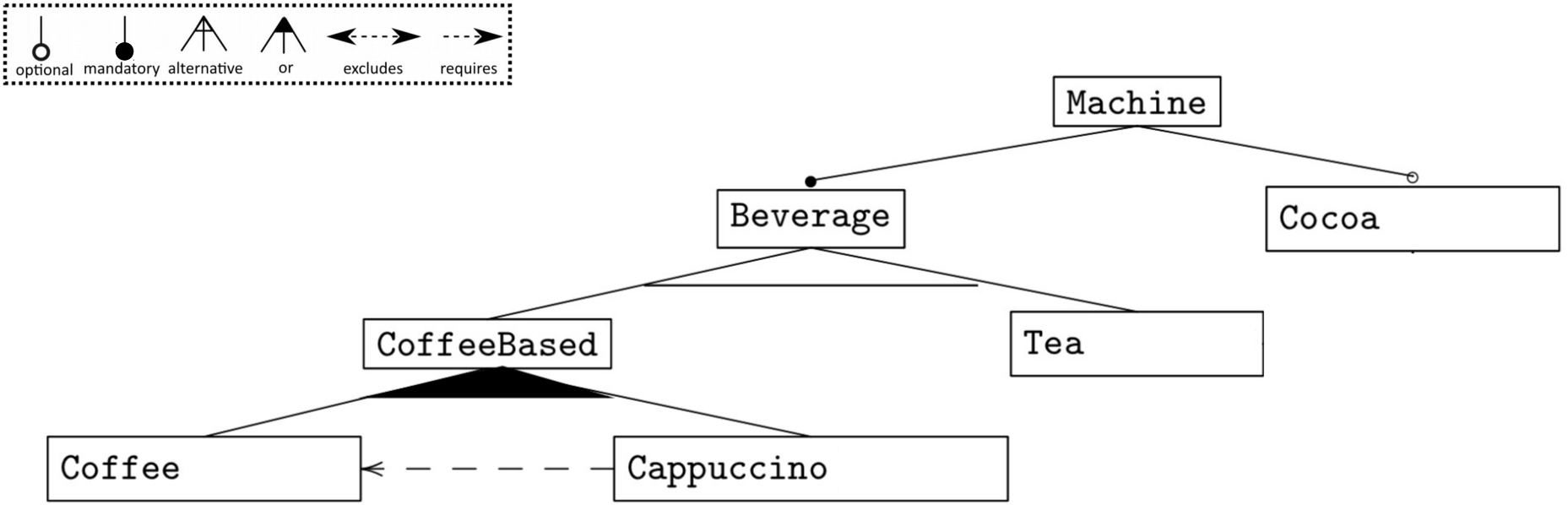
## The feature model



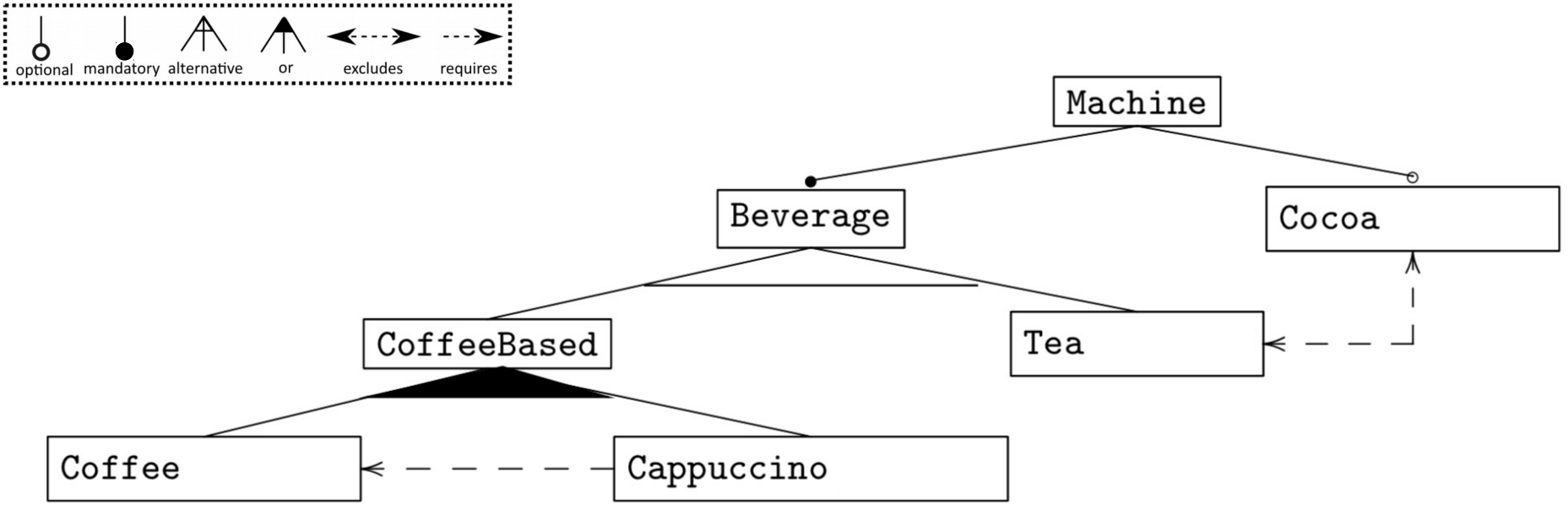
# A simple vending machine product line

## The feature model



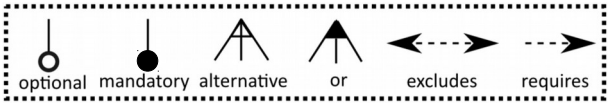
# A simple vending machine product line

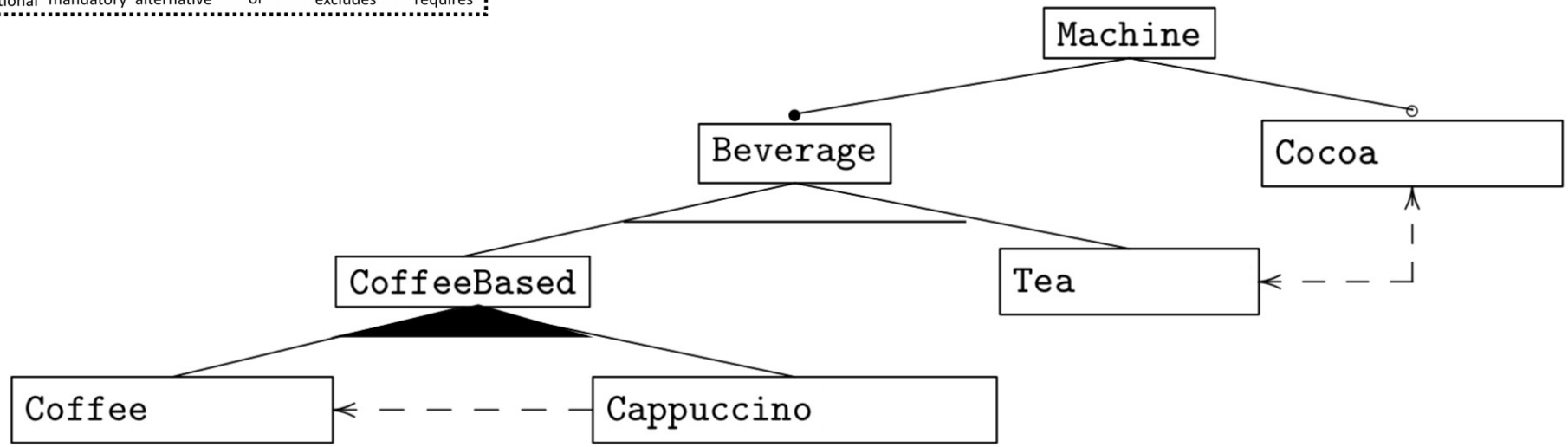
## The feature model



# A simple vending machine product line

## The feature model: Abstract & Concrete Features



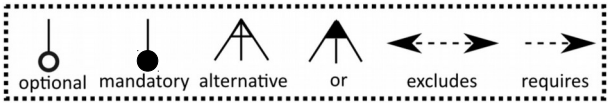



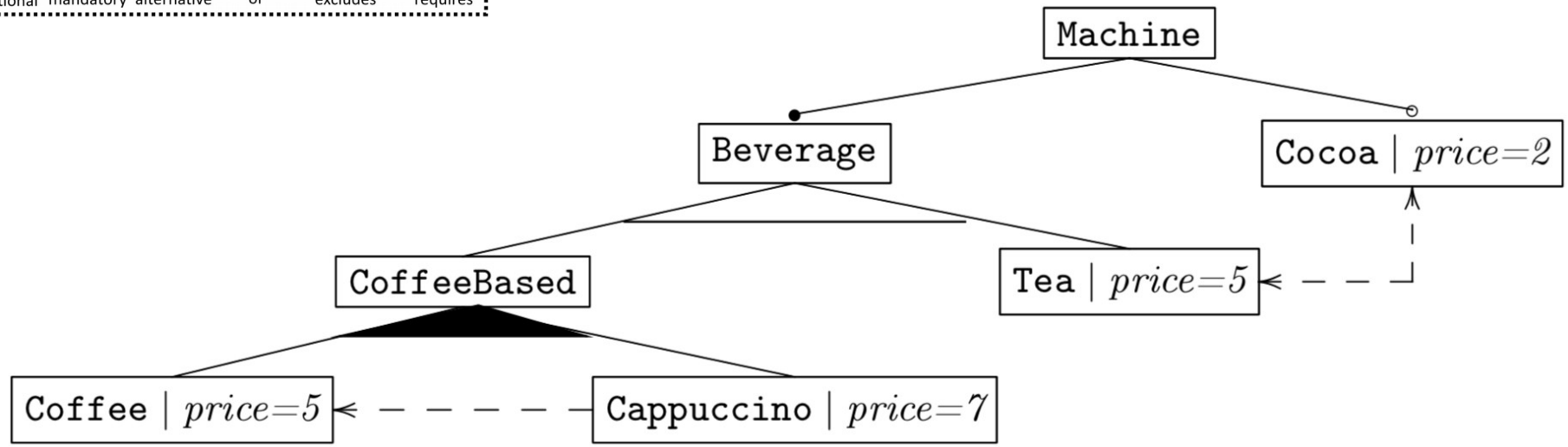

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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



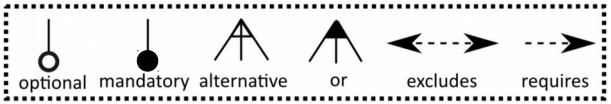



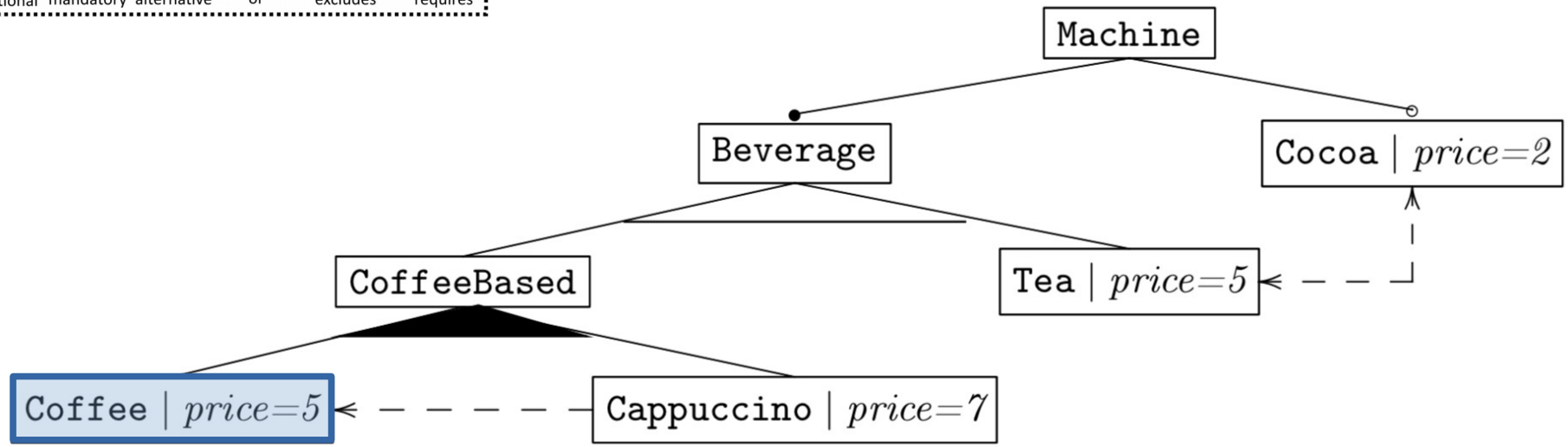

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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



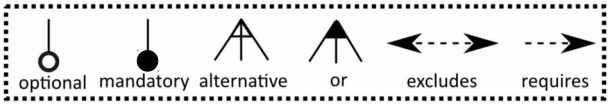




```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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





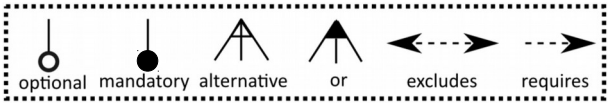


```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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





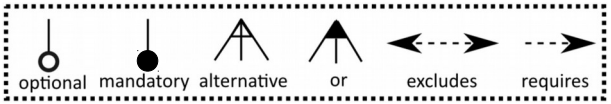


```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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





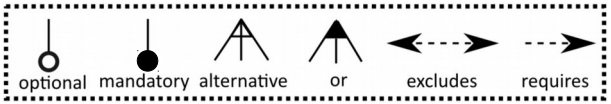


```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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





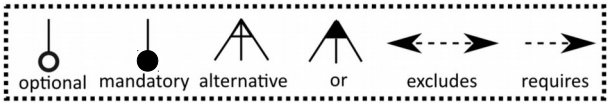


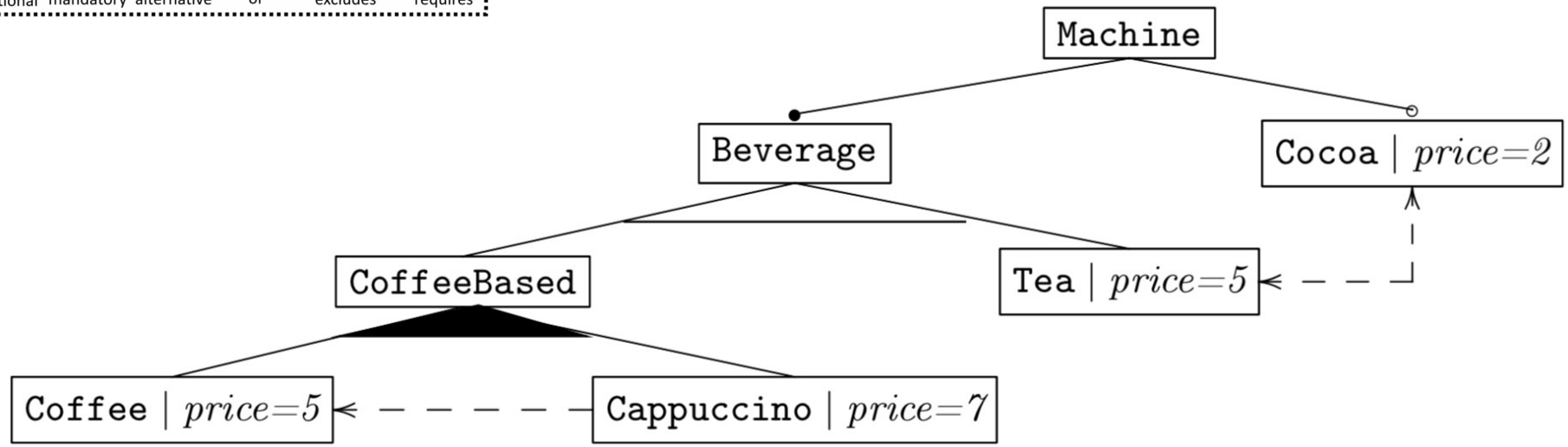
```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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 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



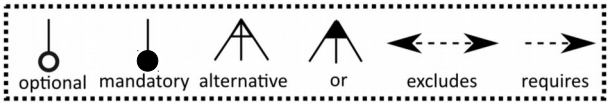



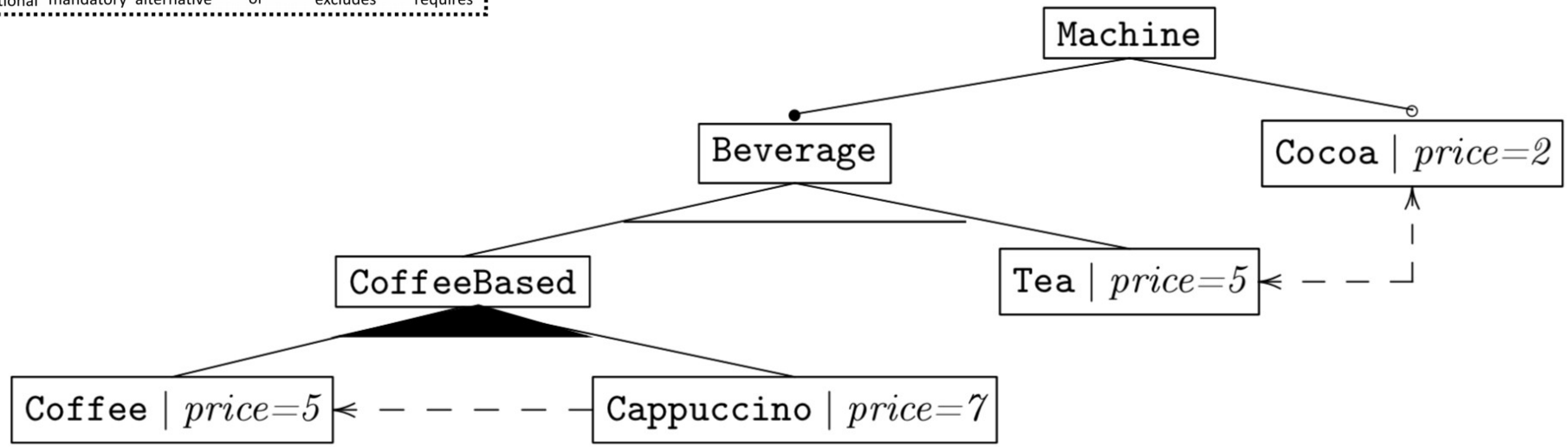

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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

# A simple vending machine product line

## The feature model: Cross-tree constraints



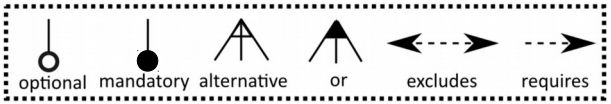



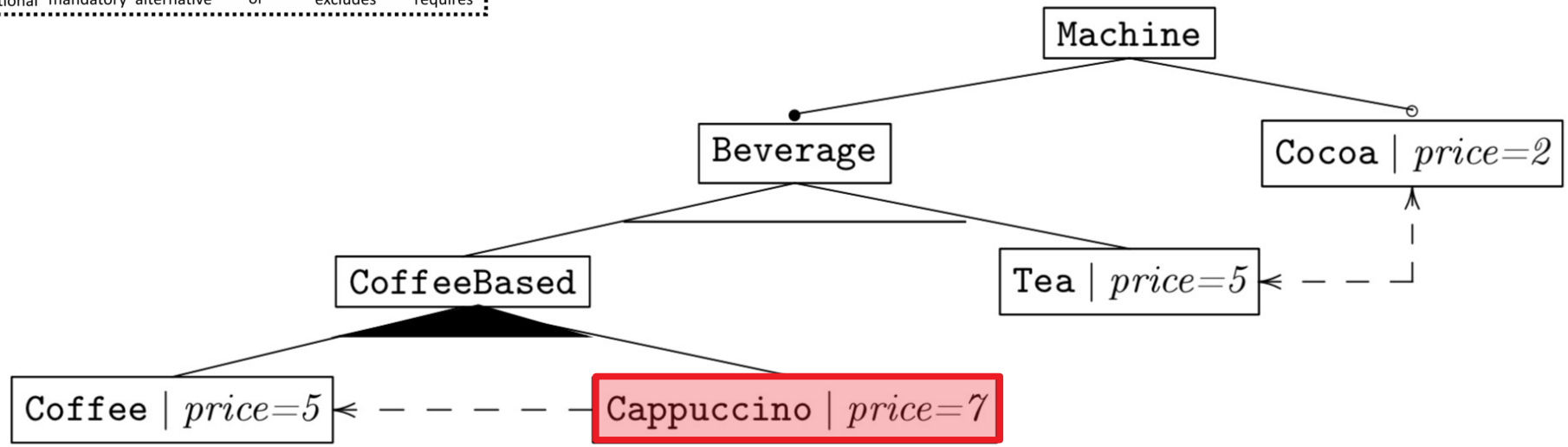

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# A simple vending machine product line

## The feature model: Cross-tree constraints



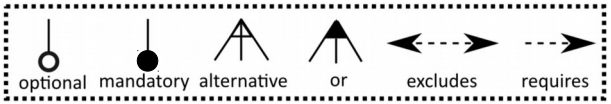



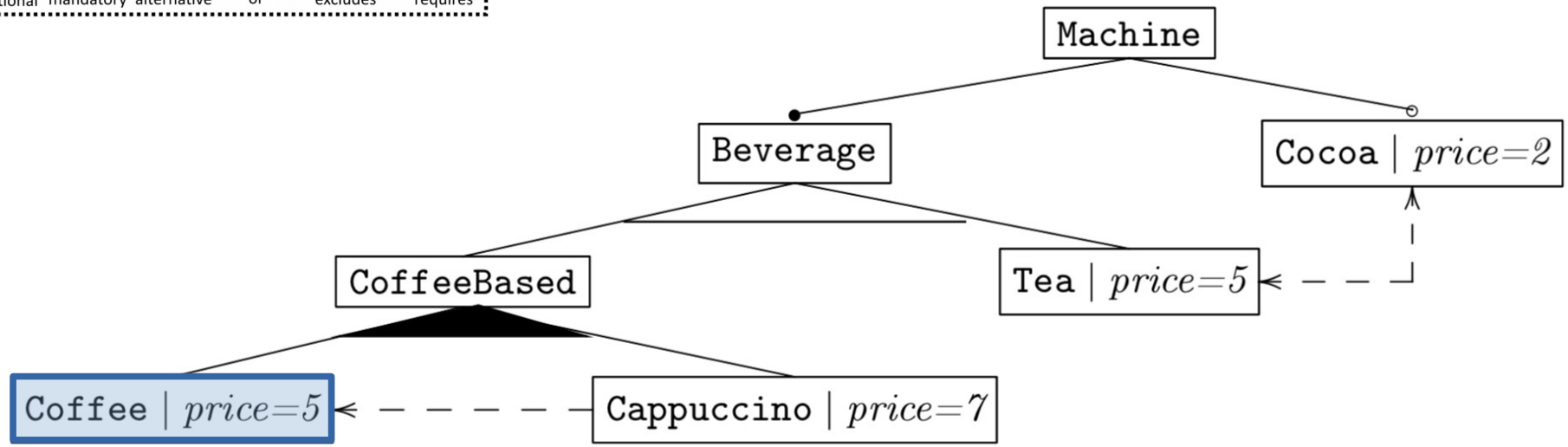

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# A simple vending machine product line

## The feature model: Cross-tree constraints



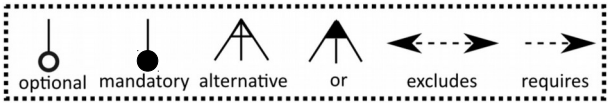



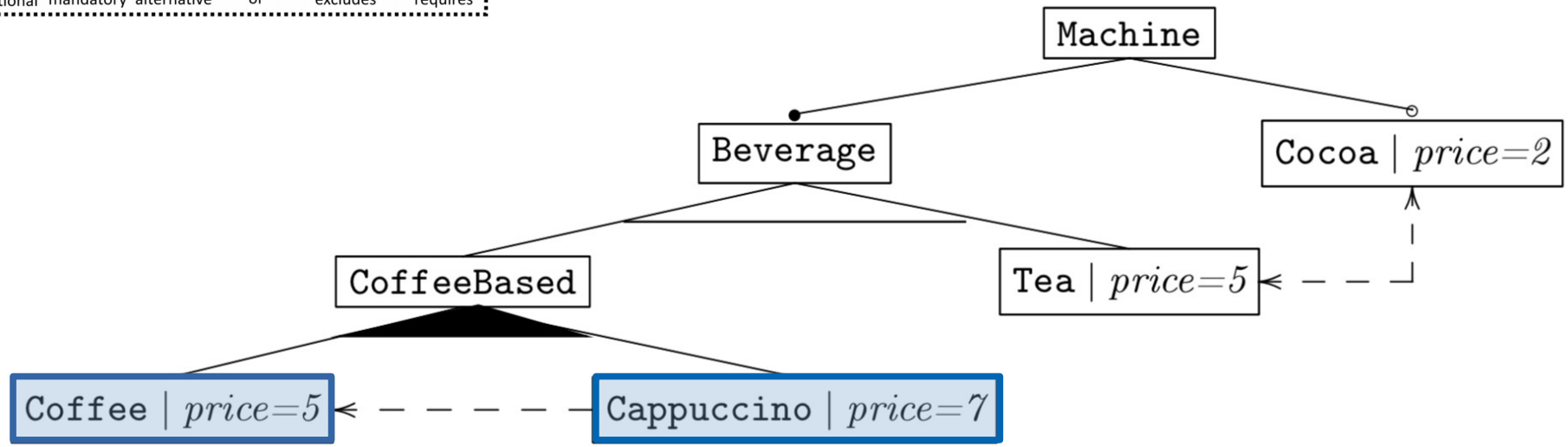

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# A simple vending machine product line

## The feature model: Cross-tree constraints



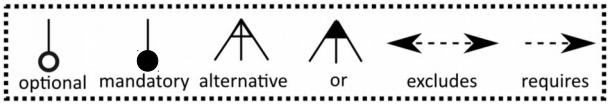



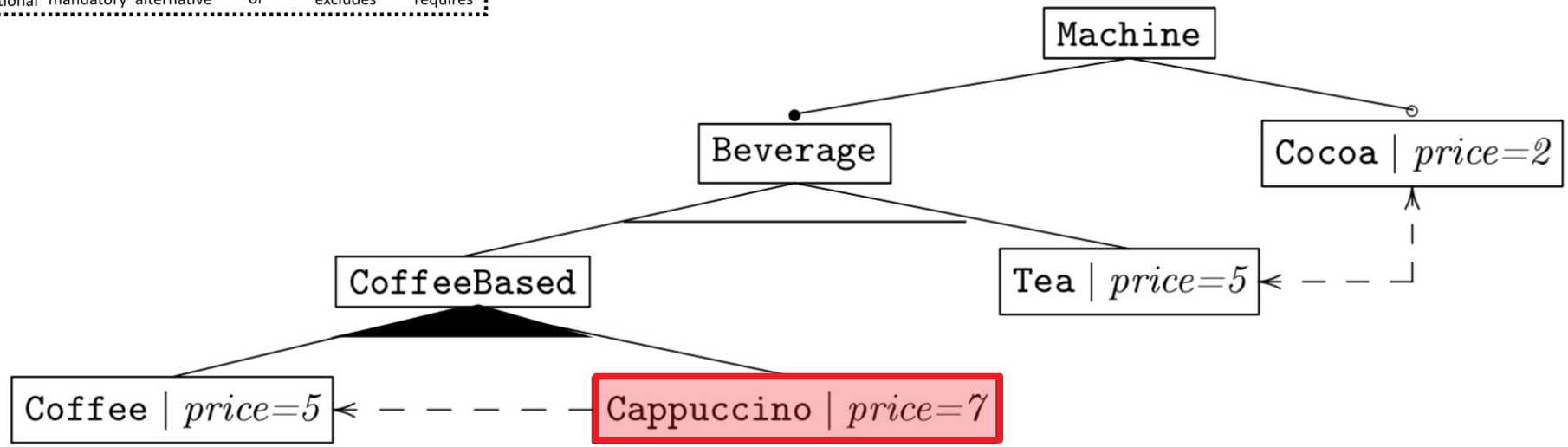

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# A simple vending machine product line

## The feature model: Cross-tree constraints



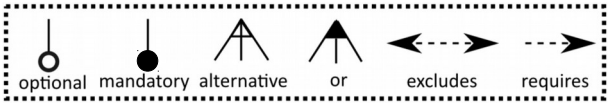



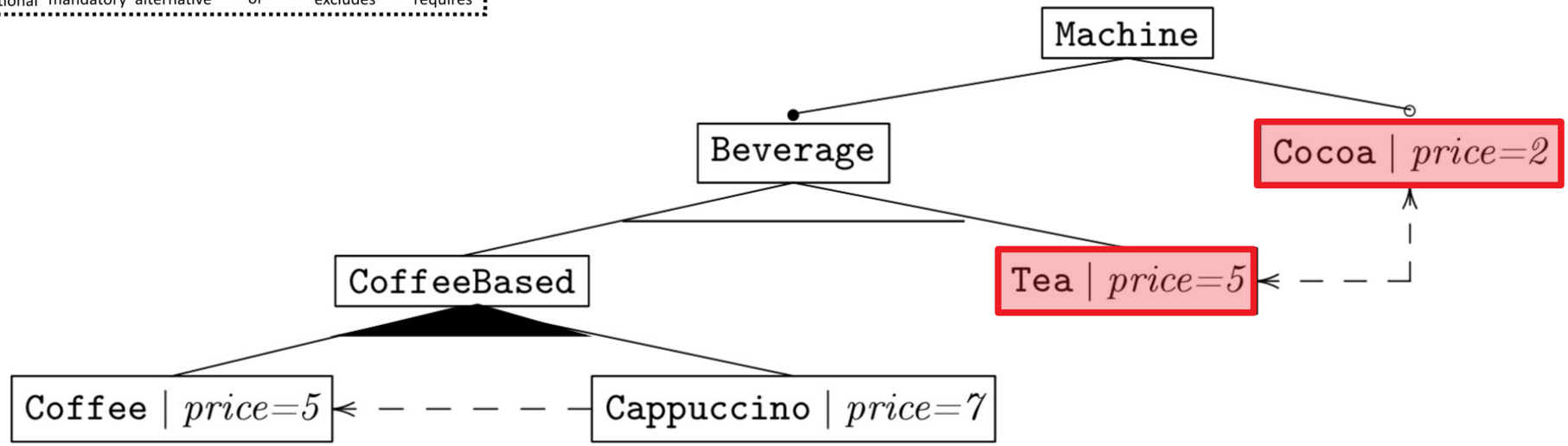

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# A simple vending machine product line

## The feature model: Cross-tree constraints



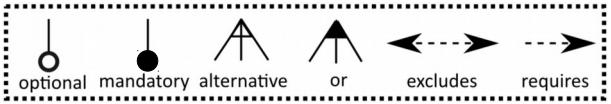



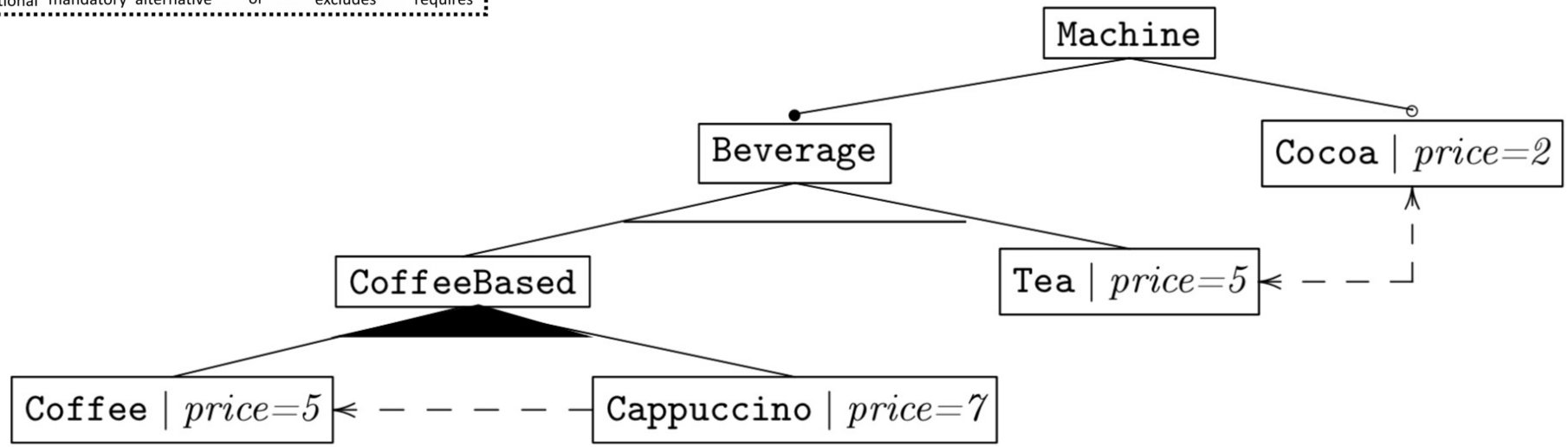

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# A simple vending machine product line

## The feature model: Quantitative constraints



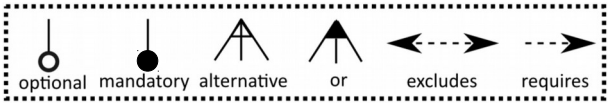



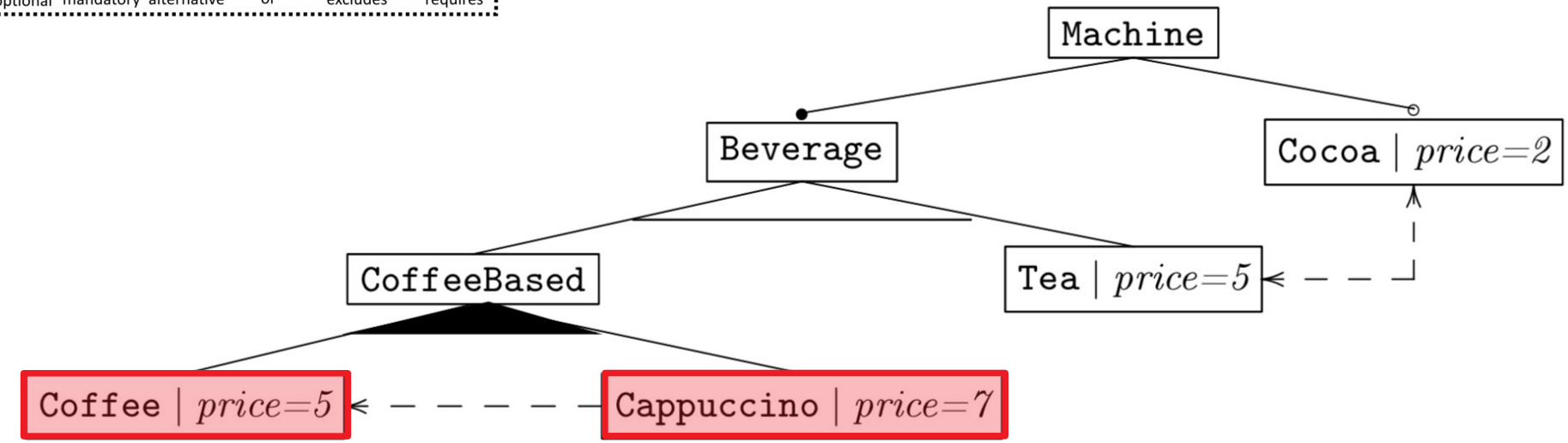

```
begin abstract features
 Machine Beverage CoffeeBased
end abstract features

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
```

# Outline

Feature Model

- Abstract and Concrete Features
- Cross-tree Constraints
- Quantitative Constraints

**Behaviour**

- Actions and Action Constraints
- Transitions
- Initial Configuration

MultiVeStA Analysis

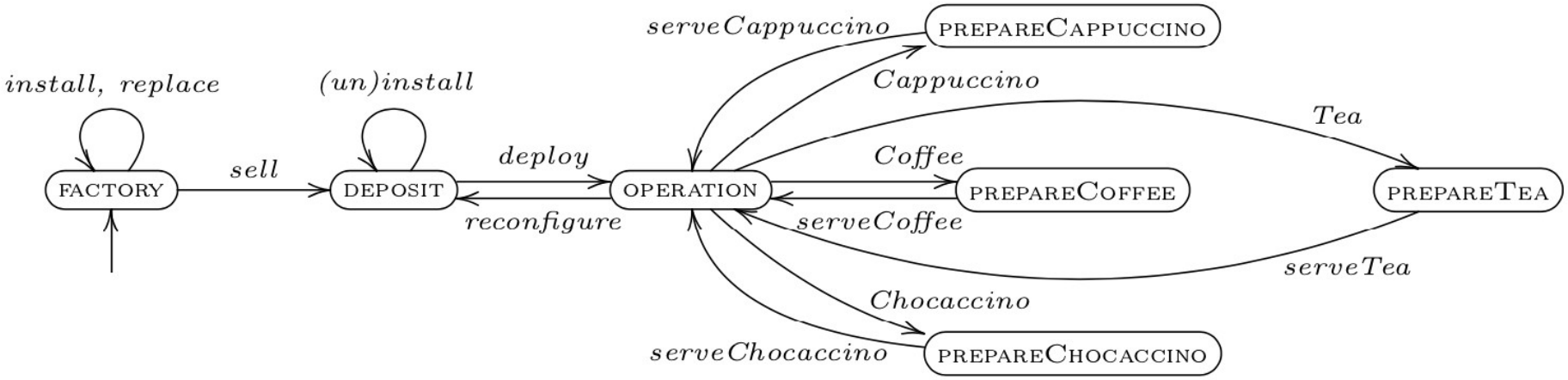
- Analysis when a condition holds
- Analysis at varying of time

An Application to a Simple Security Scenario

- Schneier's SafeLock Attack Tree

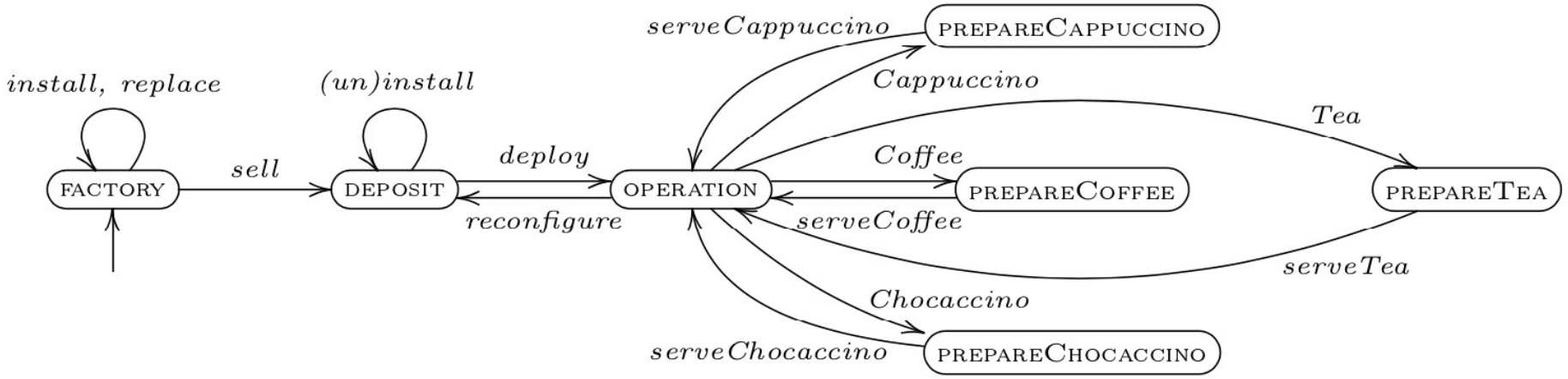
# A simple vending machine product line

## Behaviour: actions and action constraints



# A simple vending machine product line

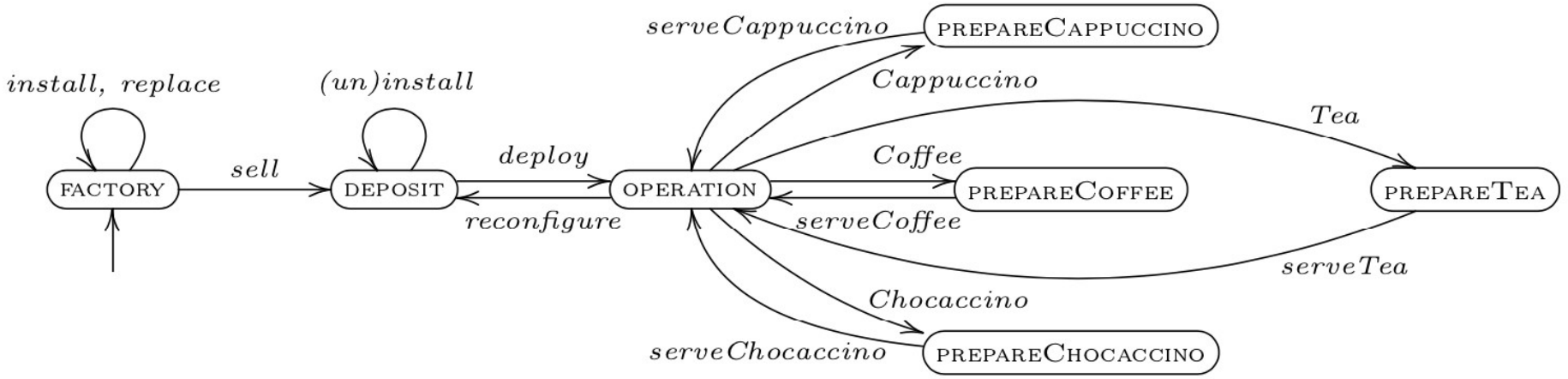
## Behaviour: actions and action constraints




```
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



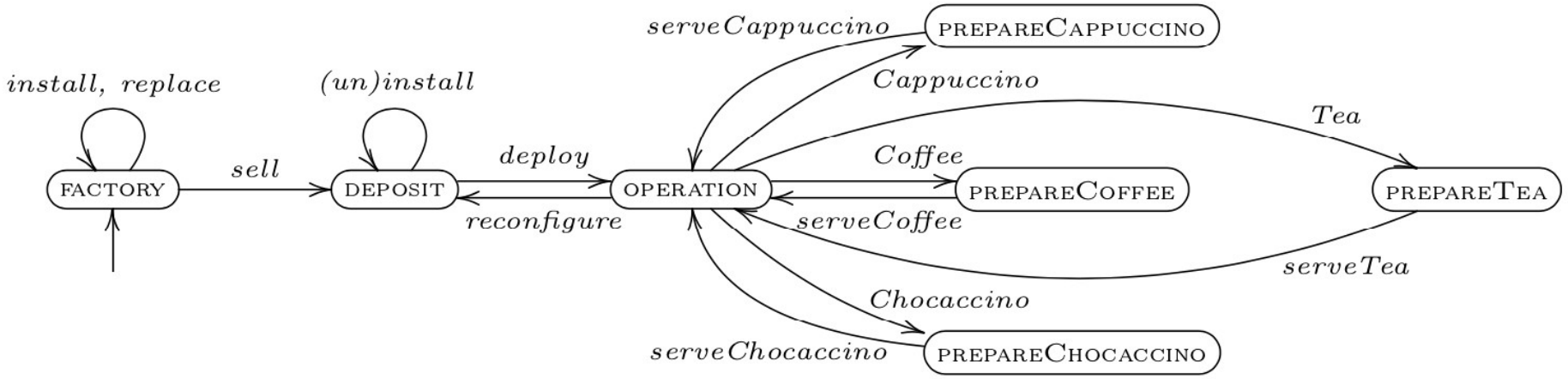

```
begin processes diagram
 begin process dynamics

 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

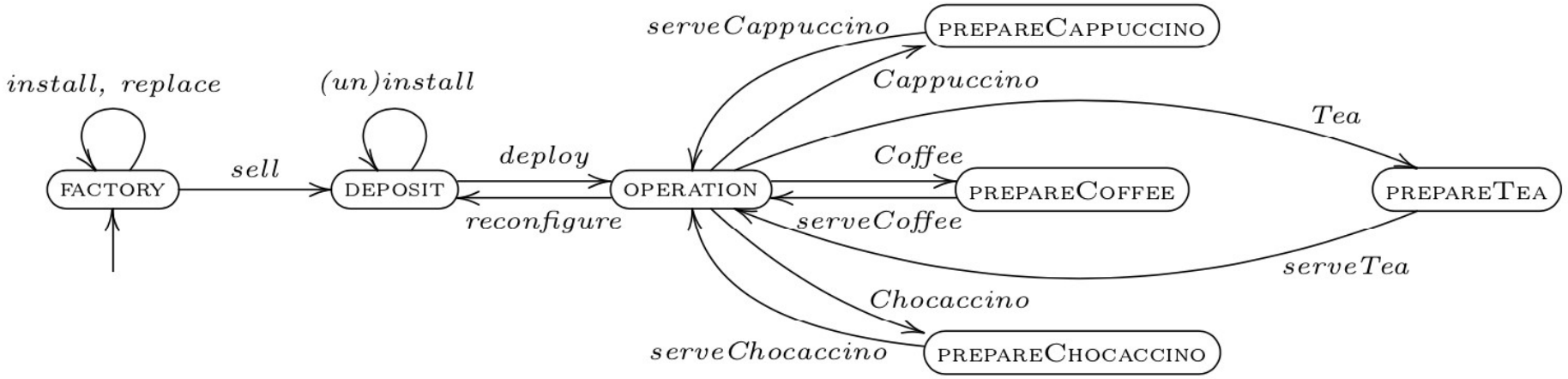



```
begin actions
 sell deploy reconfigure
 chocaccino
 serveCoffee serveCappuccino
 serveChocaccino serveTea
end actions

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

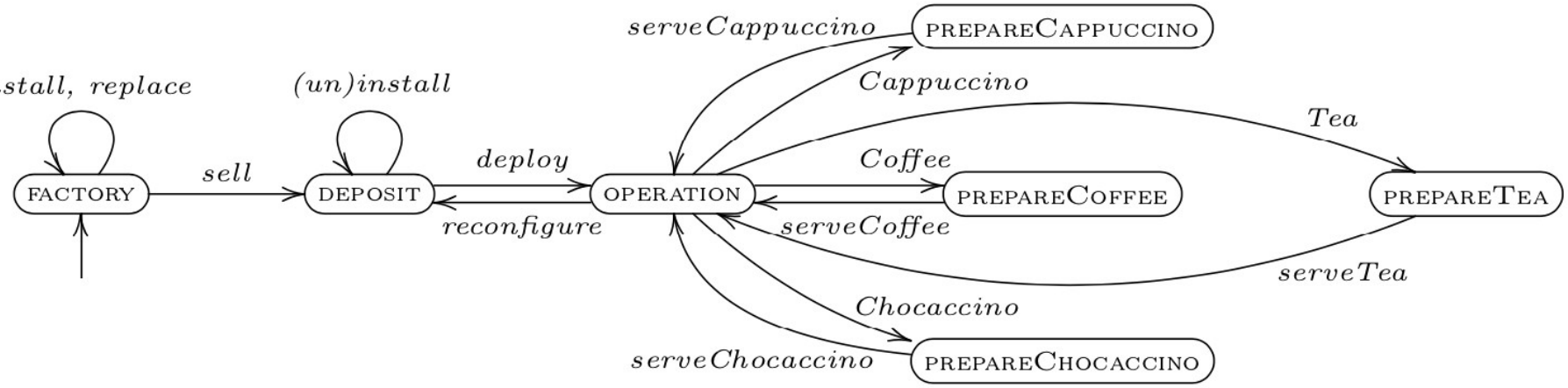



```
begin variables
 sold = 0
 deploys = 0
end variables

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

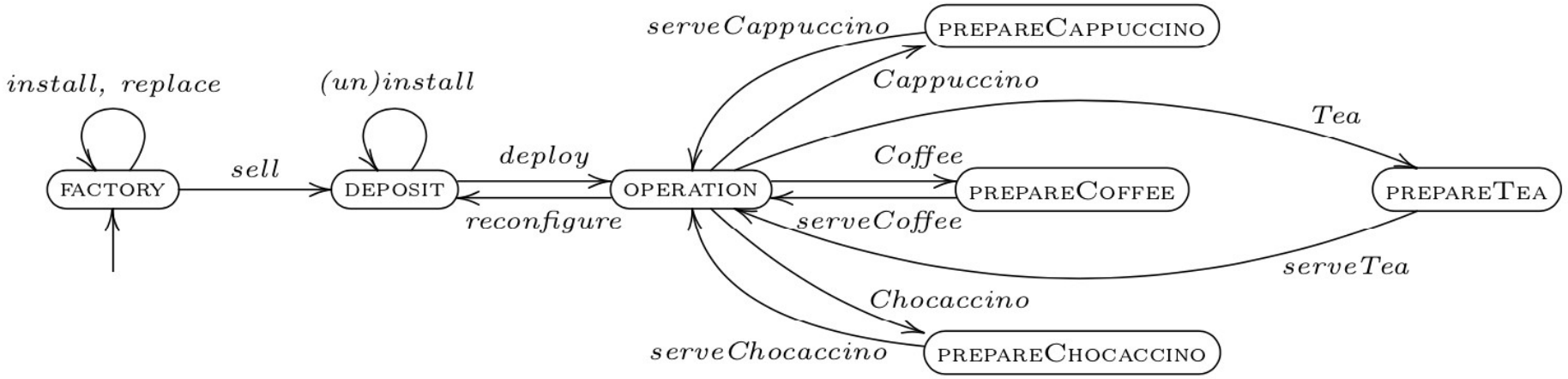



```
begin variables
 sold = 0
 deploys = 0
end variables

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
```

# A simple vending machine product line

## Behaviour: inital configuration

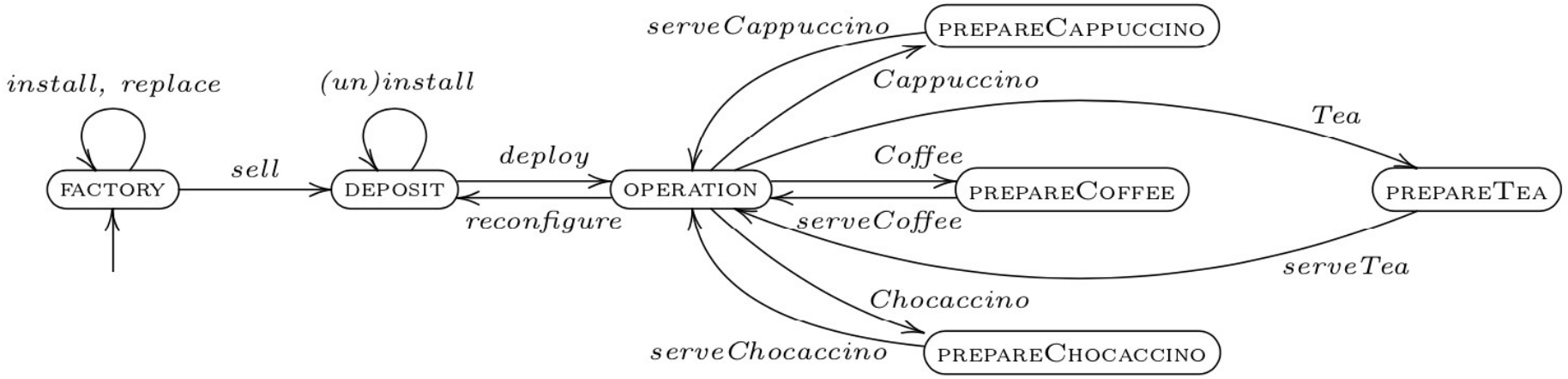



```
begin variables
 sold = 0
 deploys = 0
end variables

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 simple vending machine product line

## Behaviour: inital configuration




```
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 init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

```
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
```

# Outline

## Feature Model

- Abstract and Concrete Features
- Cross-tree Constraints
- Quantitative Constraints

## Behaviour

- Actions and Action Constraints
- Transitions
- Initial Configuration

## **MultiVeStA Analysis**

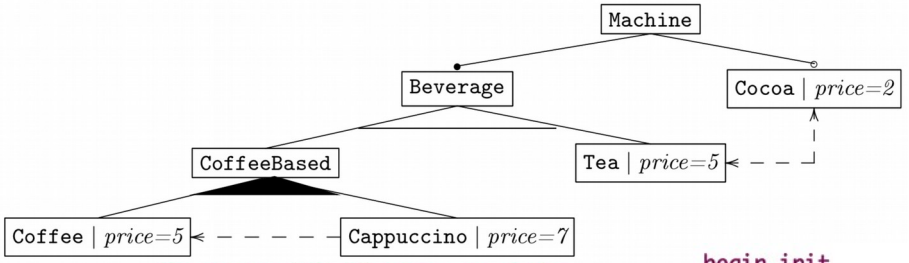
- Analysis when a condition holds
- Analysis at varying of time

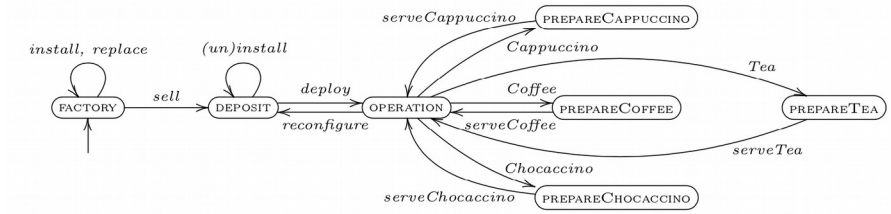
## An Application to a Simple Security Scenario

- Schneier's SafeLock Attack Tree

# A simple vending machine product line

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```

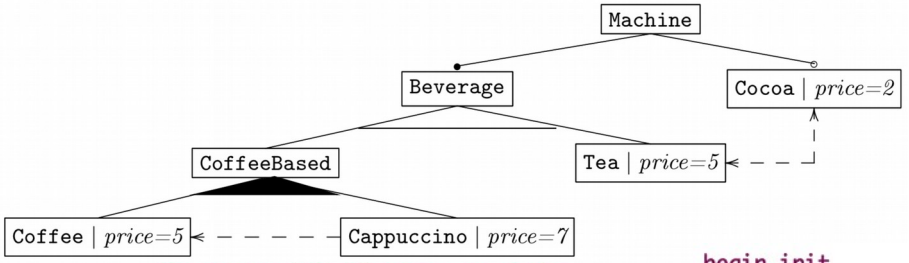
```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

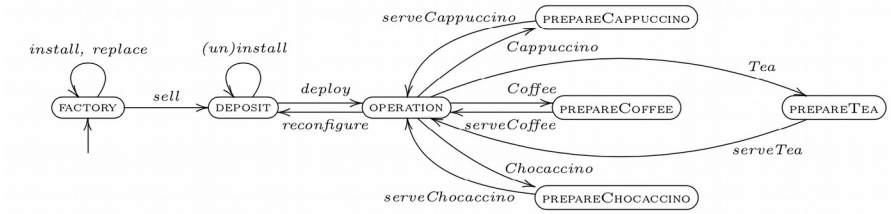
```
begin variables
 sold = 0
 deploys = 0
end variables
```

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

# A simple vending machine product line

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```

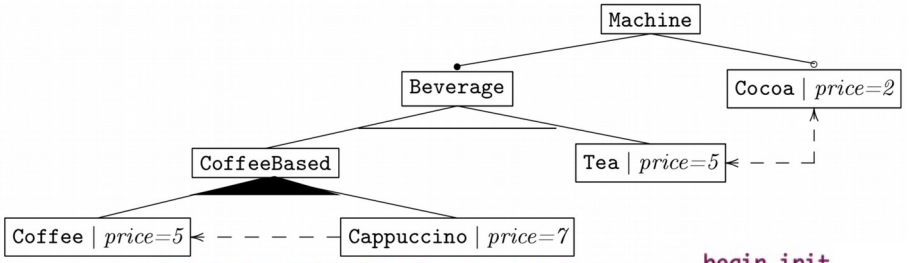
```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

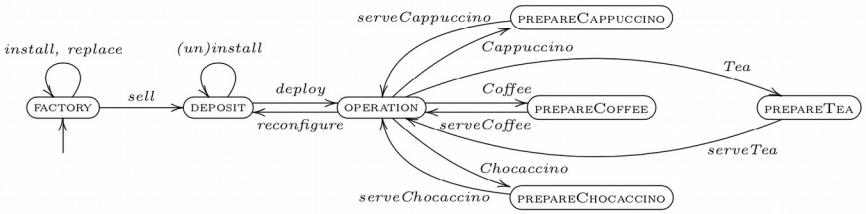
```
begin variables
 sold = 0
 deploys = 0
end variables
```

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

# A simple vending machine product line

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

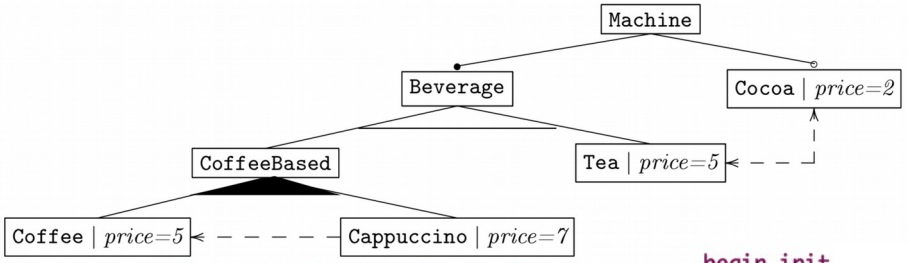
```
begin variables
 sold = 0
 deploys = 0
end variables
```

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

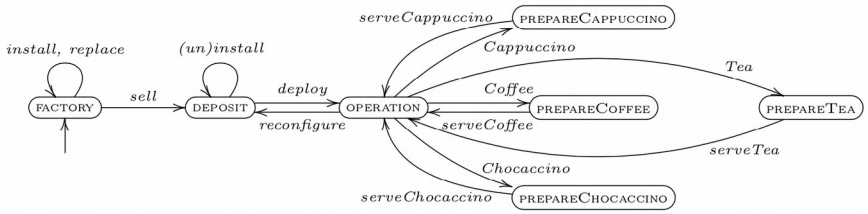
```
begin processes diagram
 begin process dynamics

 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

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```




```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

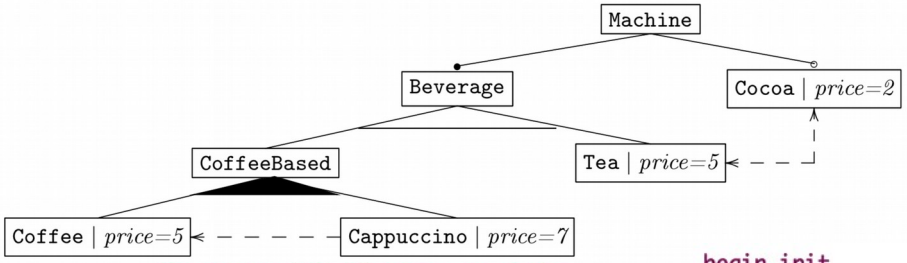
```
begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

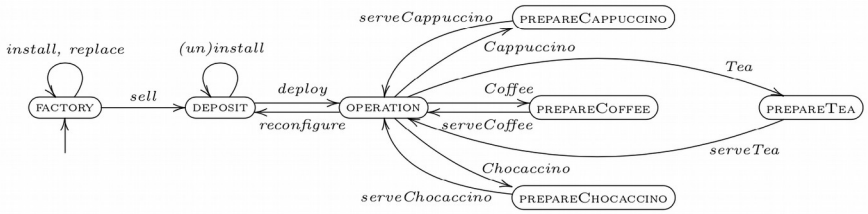
```
begin analysis

query = eval when {sold == 1.0 } :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```

```
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

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

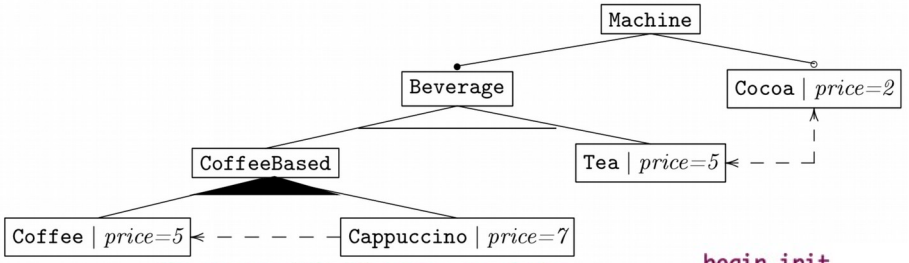
```
begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

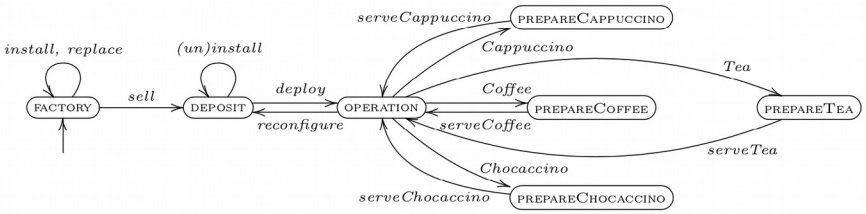
```
begin analysis

query = eval when {sold == 1.0 } :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```

| Price | Coffee | Tea | Cappuccino | Cocoa |
|---|---|---|---|---|
| 5.68 | 0.36 | 0.64 | 0.00 | 0.34 |

# A simple vending machine product line

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 10 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

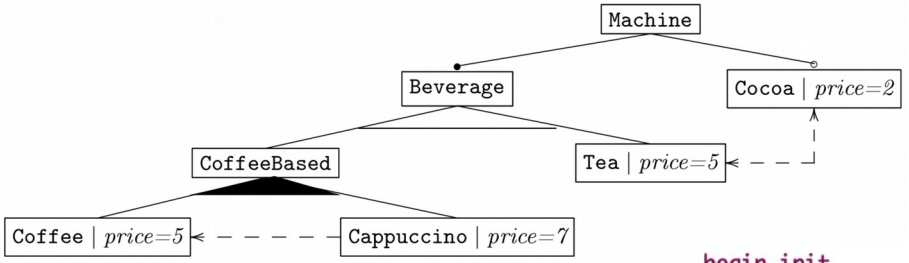
```
begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

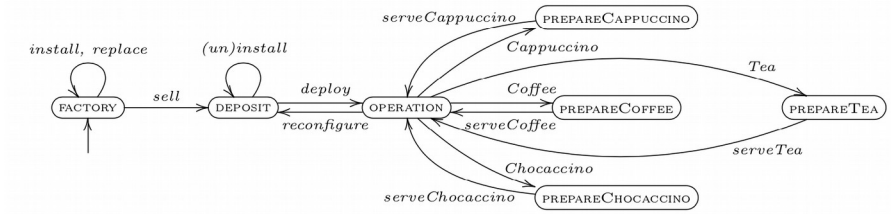
```
begin analysis

query = eval when {sold == 1.0 } :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```

| | Price | Coffee | Tea | Cappuccino | Cocoa |
|---|---|---|---|---|---|
| price(machine) <= 10 | 5.68 | 0.36 | 0.64 | 0.00 | 0.34 |

# A simple vending machine product line

## MultiVeStA Analysis: analysis of sold machines




```
begin quantitative constraints
 { price(Machine) <= 15 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

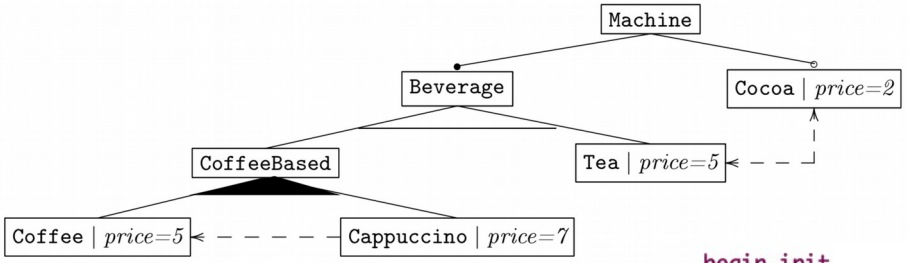
```
begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

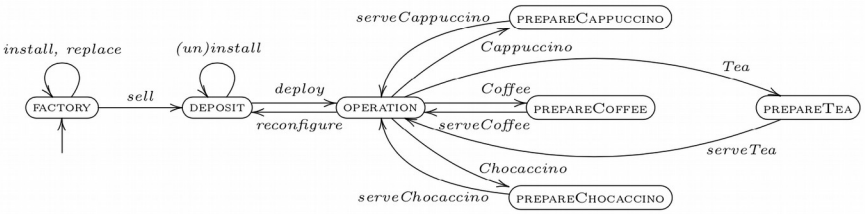
```
begin analysis

query = eval when {sold == 1.0 } :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```

| | Price | Coffee | Tea | Cappuccino | Cocoa |
|---|---|---|---|---|---|
| price(machine) <= 10 | 5.68 | 0.36 | 0.64 | 0.00 | 0.34 |
| price(machine) <= 15 | 9.07 | 0.49 | 0.51 | 0.45 | 0.44 |

# A simple vending machine product line

## MultiVeStA Analysis: analysis at varying of time




```
begin quantitative constraints
 { price(Machine) <= 15 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

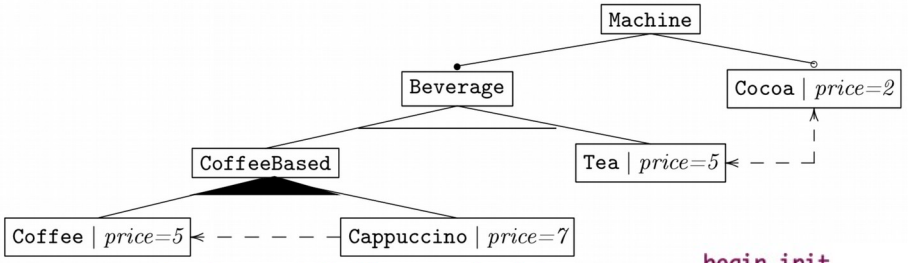
```
begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

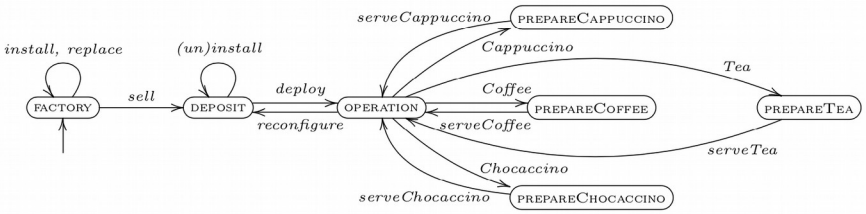
```
begin analysis

query = eval when {sold == 1.0 } :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```

| | Price | Coffee | Tea | Cappuccino | Cocoa |
|---|---|---|---|---|---|
| price(machine) <= 10 | 5.68 | 0.36 | 0.64 | 0.00 | 0.34 |
| price(machine) <= 15 | 9.07 | 0.49 | 0.51 | 0.45 | 0.44 |

# A simple vending machine product line

## MultiVeStA Analysis: analysis at varying of time




```
begin quantitative constraints
 { price(Machine) <= 15 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

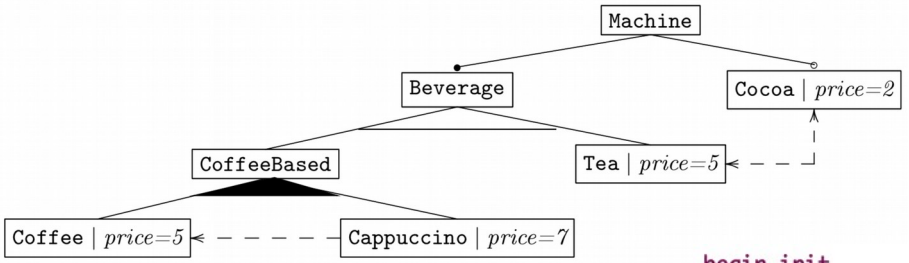
```
begin action constraints
 do(chocaccino) -> (has(Cappuccino)
                    and has(Cocoa) )
end action constraints
```

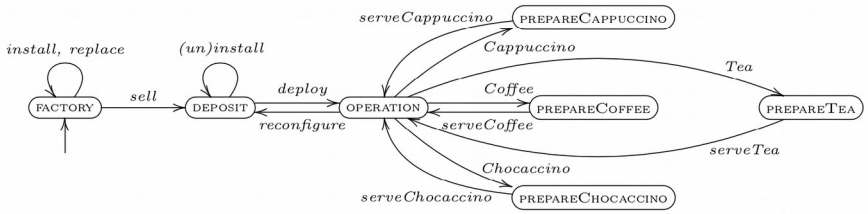
```
begin analysis

query = eval from 0 to 60 by 1 :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```

| | Price | Coffee | Tea | Cappuccino | Cocoa |
|---|---|---|---|---|---|
| price(machine) <= 10 | 5.68 | 0.36 | 0.64 | 0.00 | 0.34 |
| price(machine) <= 15 | 9.07 | 0.49 | 0.51 | 0.45 | 0.44 |

# A simple vending machine product line

## MultiVeStA Analysis: analysis at varying of time



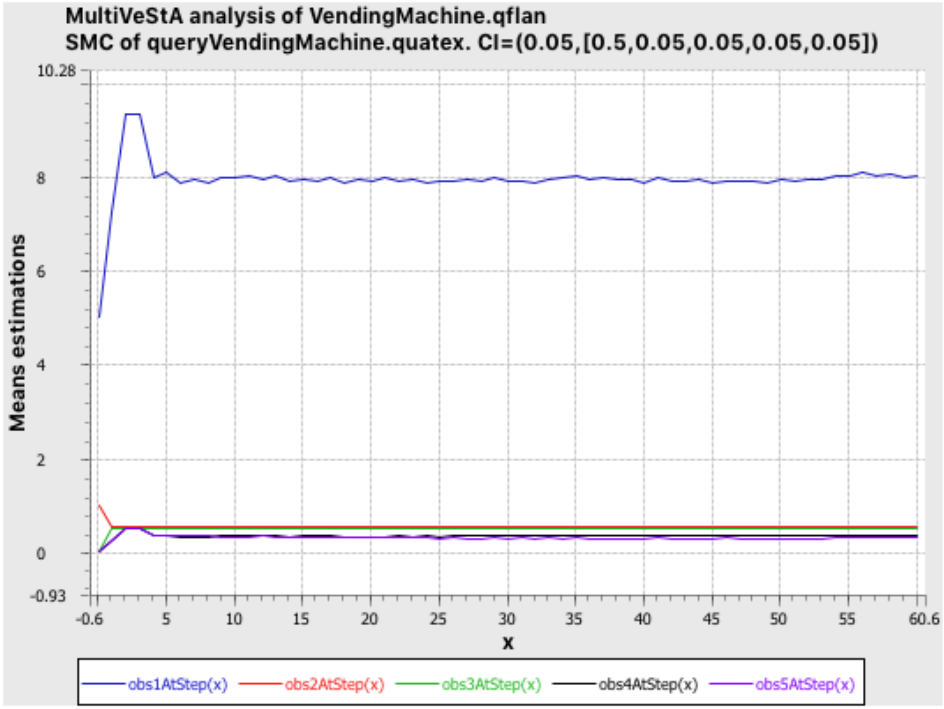

```
begin quantitative constraints
 { price(Machine) <= 15 }
end quantitative constraints
```

```
begin init
 installedFeatures = { Coffee }
 initialProcesses = dynamics
end init
```

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

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

```
begin analysis

query = eval from 0 to 60 by 1 :
{  price(Machine) [delta=0.5],
   Coffee , Tea , Cappuccino , Cocoa
}

default delta=0.05
alpha = 0.05
parallelism = 1

end analysis
```



# Outline

## Feature Model

- Abstract and Concrete Features
- Cross-tree Constraints
- Quantitative Constraints

## Behaviour

- Actions and Action Constraints
- Transitions
- Initial Configuration

## MultiVeStA Analysis

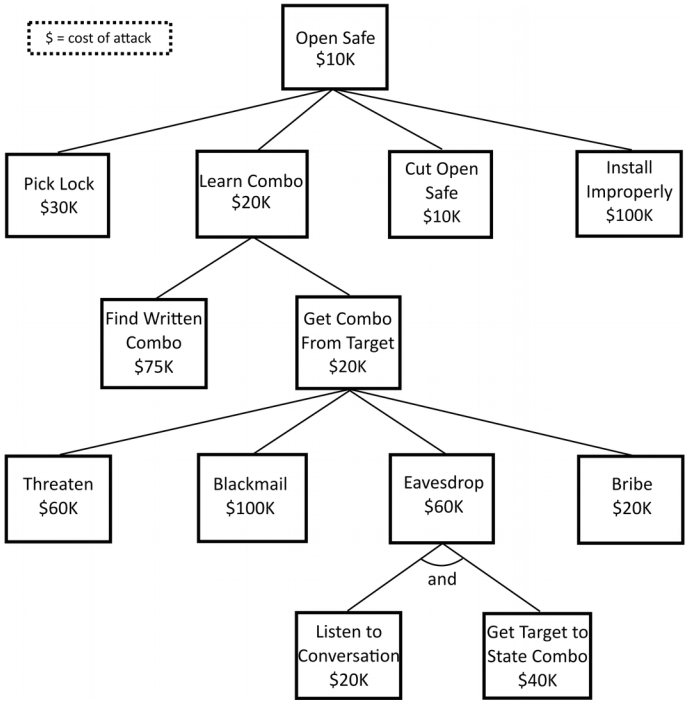
- Analysis when a condition holds
- Analysis at varying of time

## **An Application to a Simple Security Scenario**

- Schneier's SafeLock Attack Tree

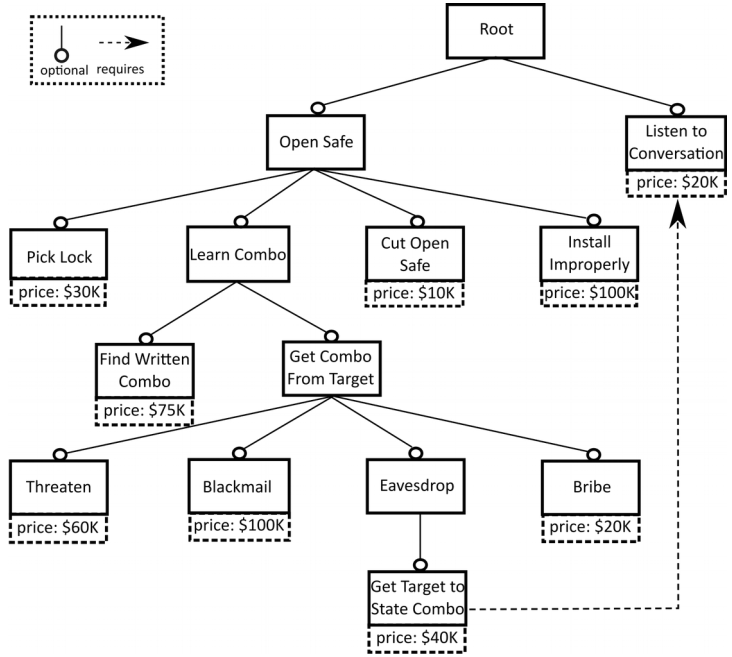
# Schneier's SafeLock Attack Tree

## An application of QFLan to security




Schneier's simple attack tree

www.schneier.com/academic/archives/1999/12/attack_trees.html

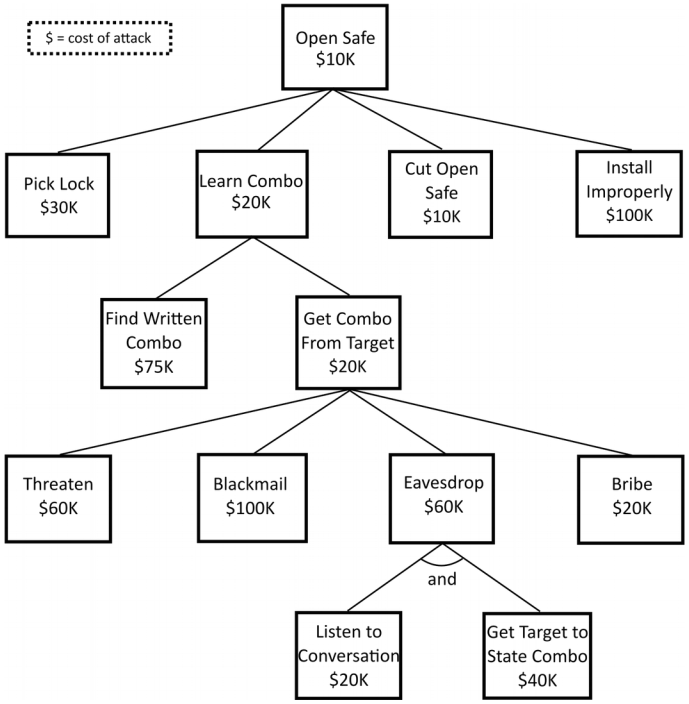



A feature model version of the attack tree

**[TSE'18]**

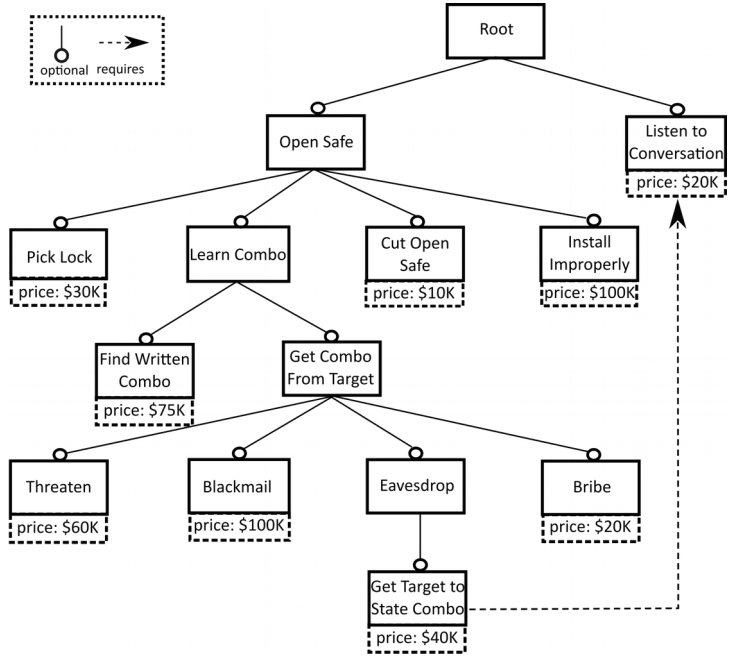
# Schneier's SafeLock Attack Tree

## An application of QFLan to security

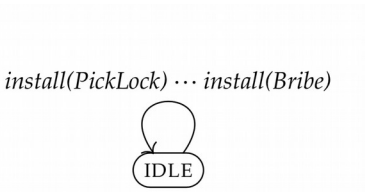



Schneier's simple attack tree

www.schneier.com/academic/archives/1999/12/attack_trees.html



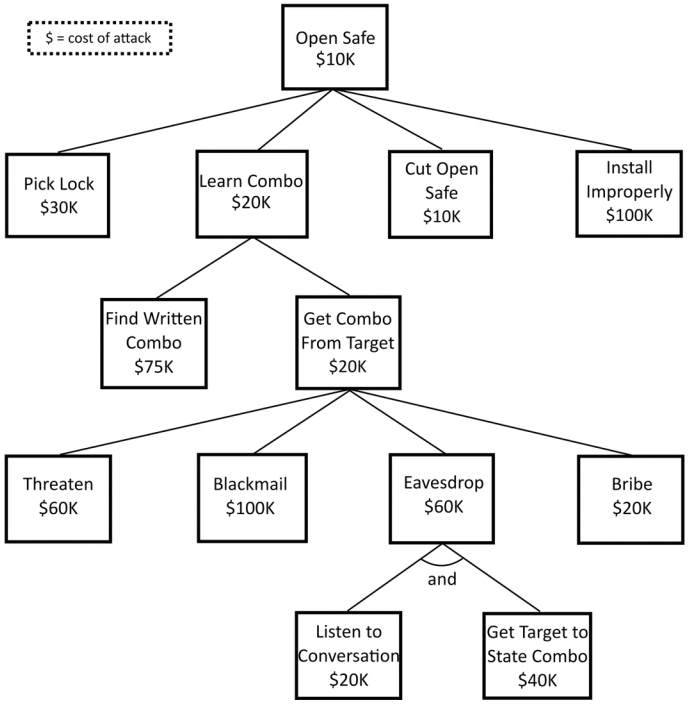

A feature model version of the attack tree **[TSE'18]**




```
begin quantitative constraints
 //Restrict to attacks that cost less than 100
 { cost(Root) <= 100 }



end quantitative constraints
```

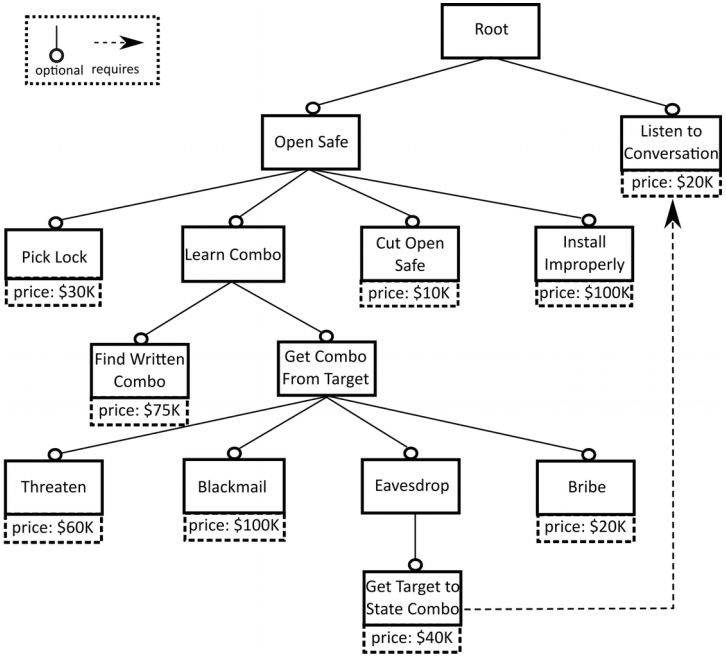
# Schneier's SafeLock Attack Tree

## An application of QFLan to security

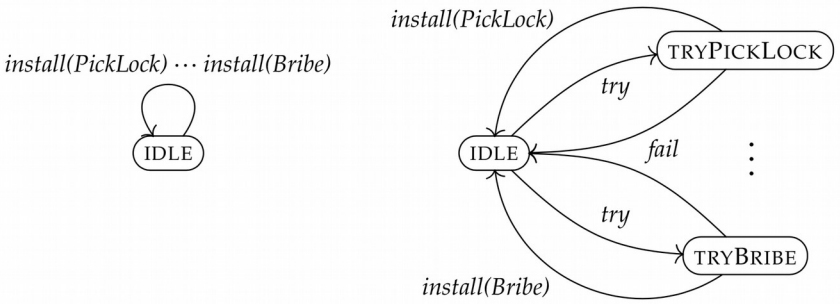



Schneier's simple attack tree

www.schneier.com/academic/archives/1999/12/attack_trees.html



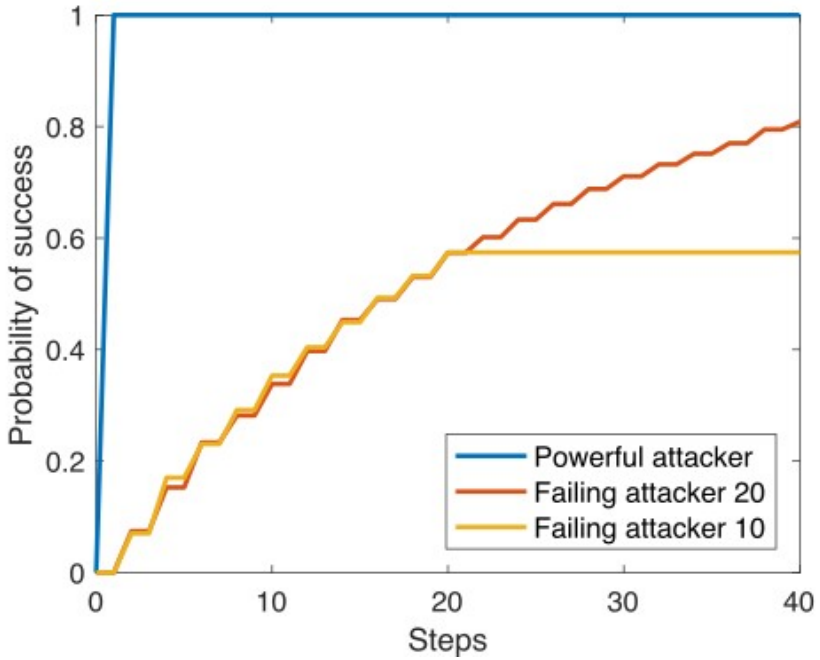

A feature model version of the attack tree **[TSE'18]**



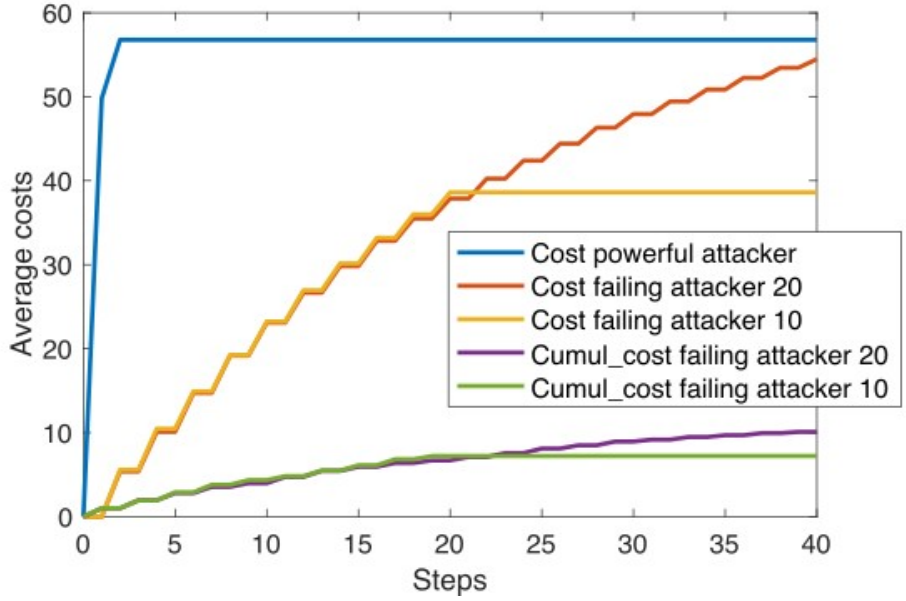

```
begin quantitative constraints
 //Restrict to attacks that cost less than 100
 { cost(Root) <= 100 }
 //Attacks can fail. Attacks attempts cost.
 //Restrict to attackers with a maximum bugdet.
 { accumulated_cost <= 10}
 //{ accumulated_cost <= 20}
end quantitative constraints
```

# Schneier's SafeLock Attack Tree

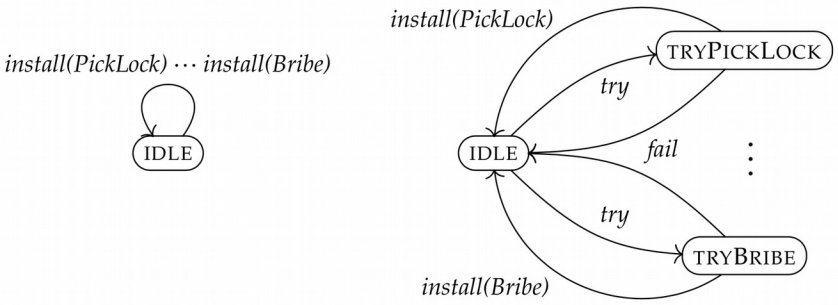
An application of QFLan to security




Probabilities of successful attacks




Costs of successful attacks




```
begin quantitative constraints
 //Restrict to attacks that cost less than 100
 { cost(Root) <= 100 }
 //Attacks can fail. Attacks attempts cost.
 //Restrict to attackers with a maximum bugdet.
 { accumulated_cost <= 10}
 //{ accumulated_cost <= 20}
end quantitative constraints
```

# Ongoing and future work

## Continue investigating applicability to security domain

- Adapt QFLan to attack trees domain
  - Defense/countermeasure nodes, defense effectiveness, attack detection rate...
- Feasibility investigated in an MSc thesis at DTU
- Papers and prototype tool under preparation

## Relate the underlying theoretical model with MTS/FTS

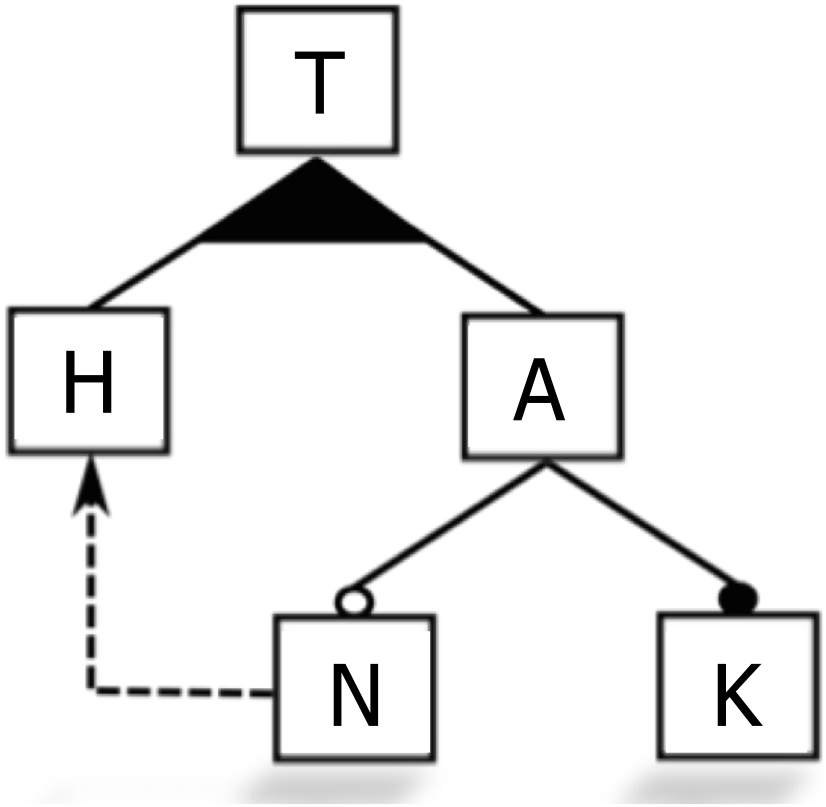
## Extend semantics with explicit notion of time

- For the analysis of time-related properties

## Synthesis of constraints

- We had to relax the constraint "price(Machine) <= 10"
- Can we synthesize the 'right' constraints automatically?

# QUESTIONS?




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