

Variability Models

Dave Clarke

Uppsala University, Sweden

KU Leuven, Belgium

Overview

- ◆ Context – Variability and Software Product Line Engineering (20 minutes)
- ◆ 3 Formalisms:
 - ◆ Feature Models (30 minutes)
 - ◆ Behavioural Models: Feature Transition Systems and Feature Petri Nets (1 hour)
 - ◆ Code: Delta Modelling (1 hour)
- ◆ Conclusions (10 minutes)

Variability and Software Product Line Engineering

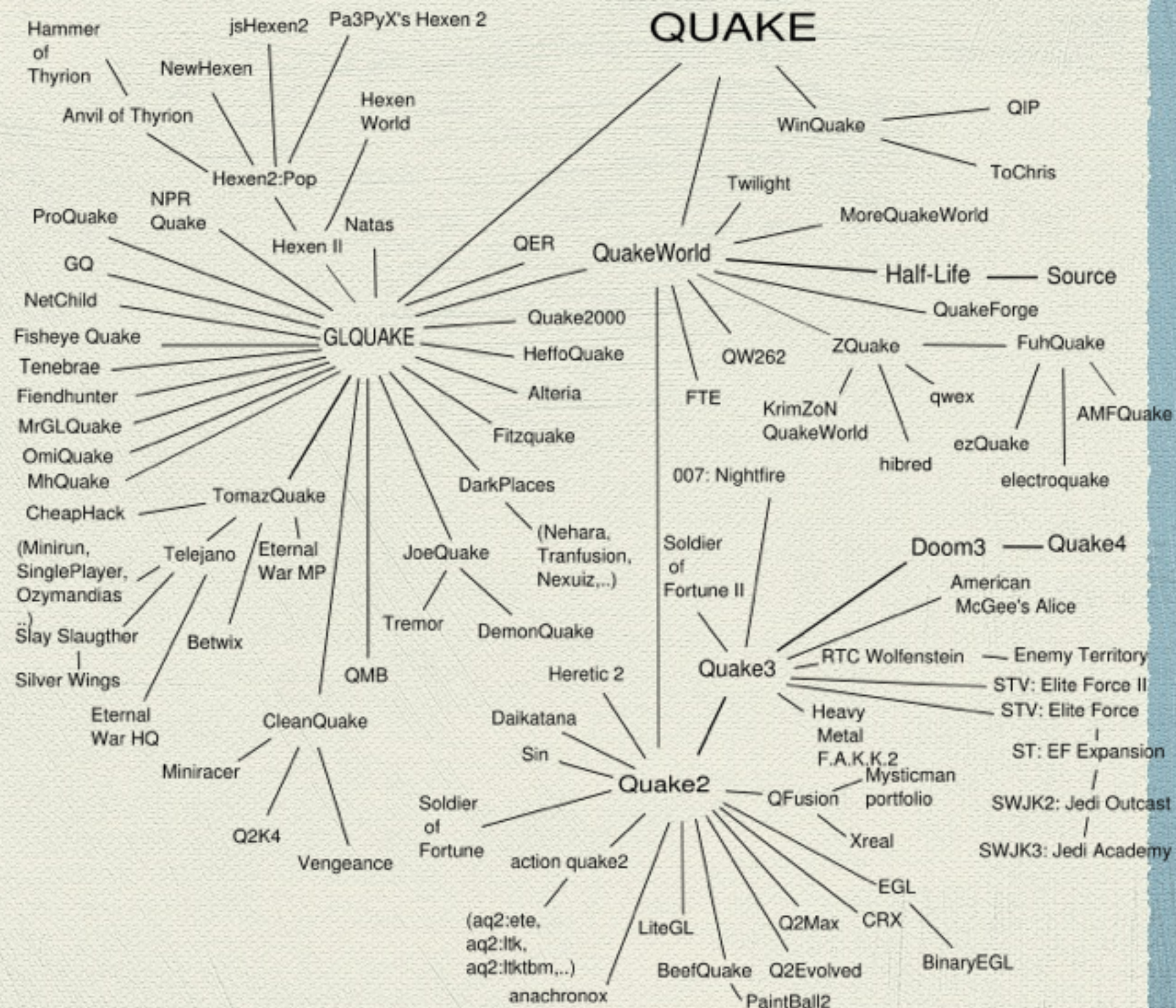
Motivation



... somehow all the same
... but a little different



... somehow all the same
... but a little different



... somehow all the same
... but a little different

Motivation: Reflection

Why are there so many products from a single company that are more or less all the same?

- ◆ Different customer requirements (market needs)
 - ◆ features vs price
- ◆ Innovation to distinguish products from competitors'
- ◆ Products sell and earn money (customers want new version).
- ◆ The company cannot afford to develop each system from scratch.

Software Product Lines

- ◆ A software product line (SPL) is a set of programs that share *significant common functionality* and structure.
- ◆ The differences between the set of programs are well-understood and organised in some form.
- ◆ Supports systematic re-use of configurable artefacts across development activities
- ◆ Product line targets specific market segment.

Software Product Line Engineering

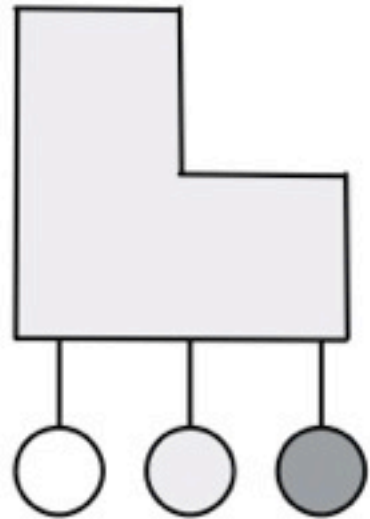
“a paradigm to develop software applications (software intensive systems and software products) using platforms and mass customisation” [Pohl et al '05]

Software Platform

a set of software building blocks with common interfaces that can be combined to derive a variety of products

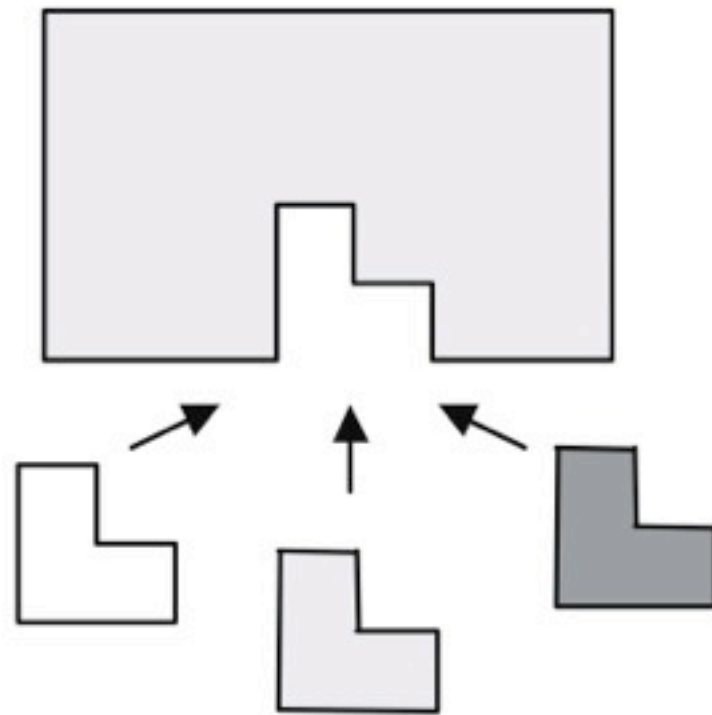
Variability Mechanisms

adaptation



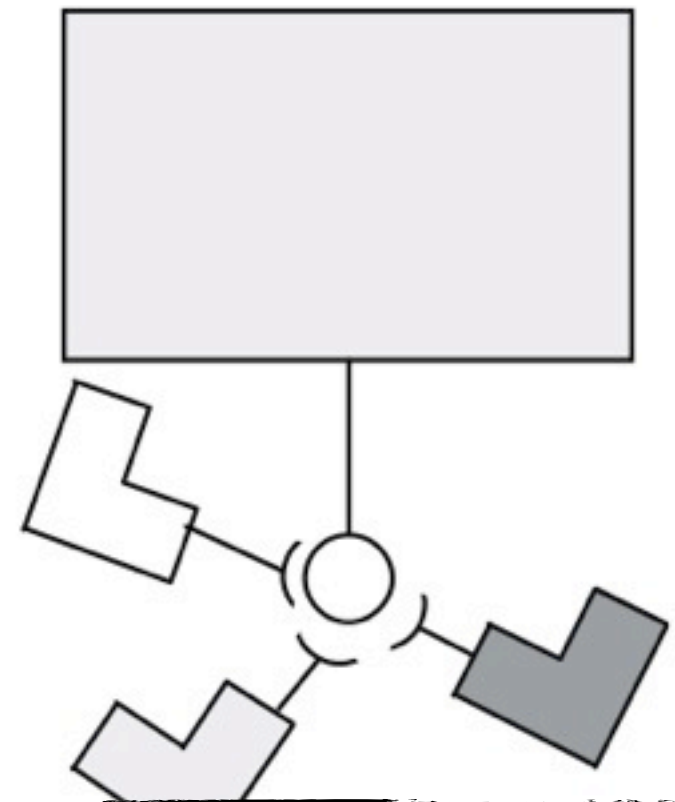
One core
implementation.
Adaptable behaviour.

replacement



Multiple Component
Implementations.
Choose one or
develop product
specific.

extension

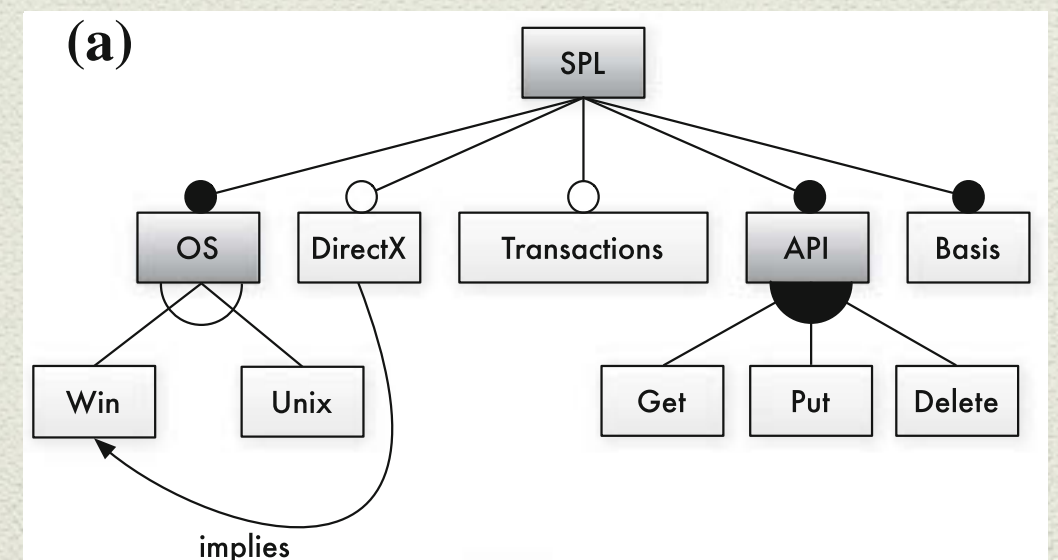


Generic Interface
for
adding
components

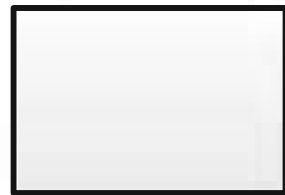
Feature Models

Feature Models

- ◆ Describe the variability of a system at a high-level of abstraction
- ◆ Expressed in terms of features (names) and sometimes attributes and cardinality

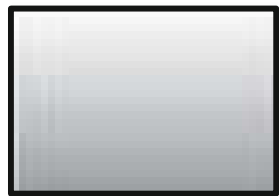


Abstract and Concrete Features



concrete

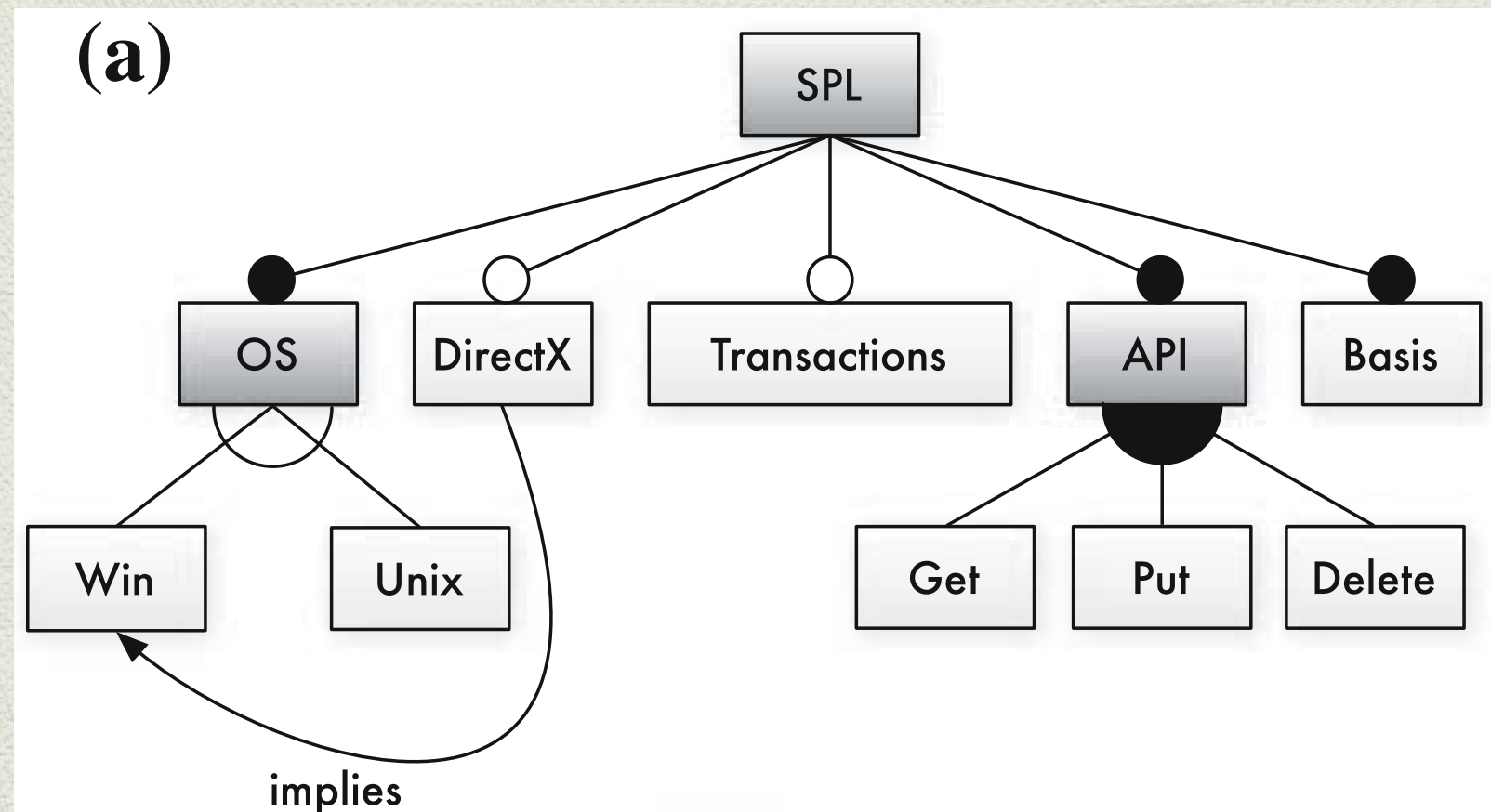
Feature of product line



abstract

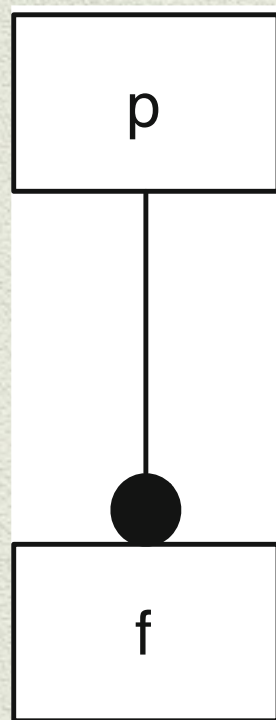
For structuring model

Abstract & Concrete Features

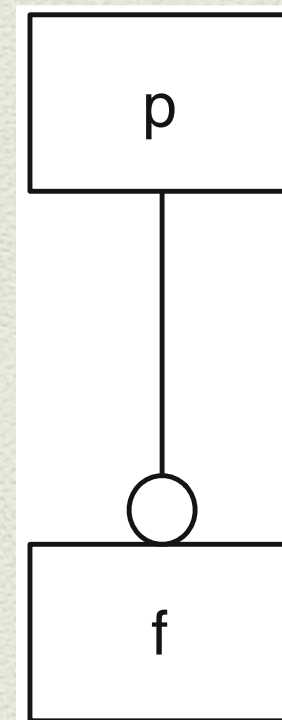


Mandatory and Optional Features

Mandatory

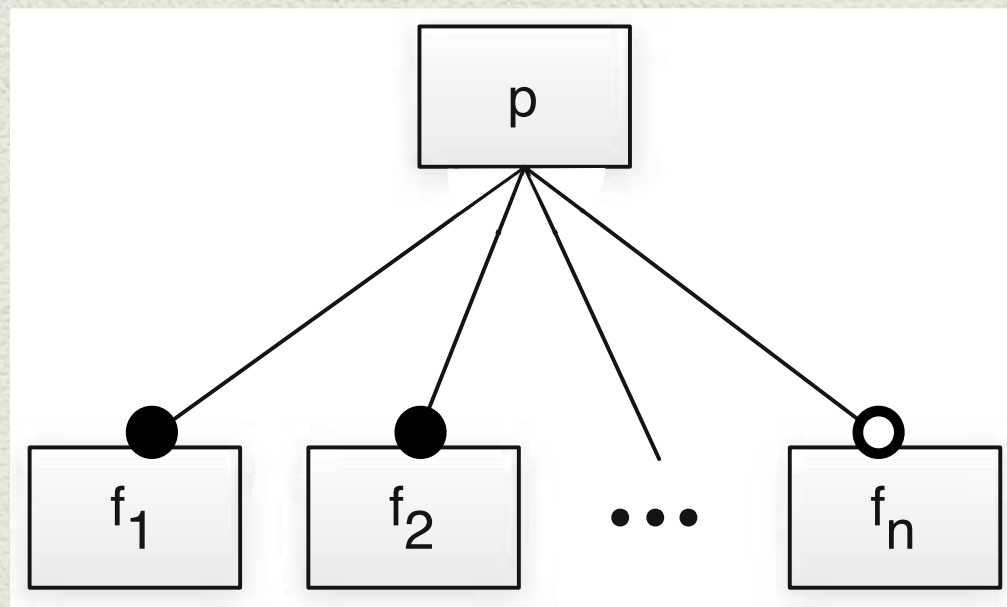


Optional

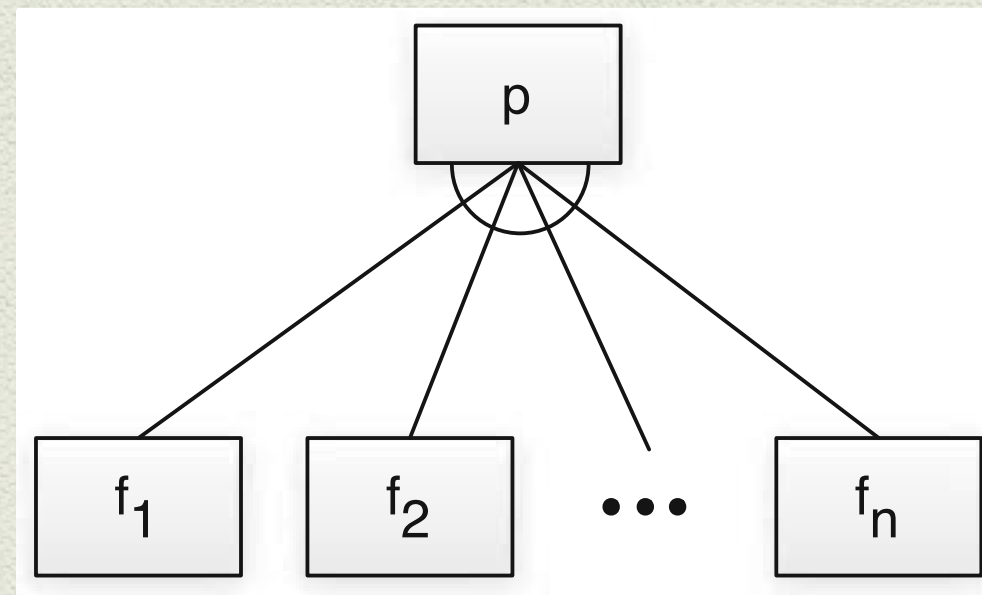


Subfeatures (AND)

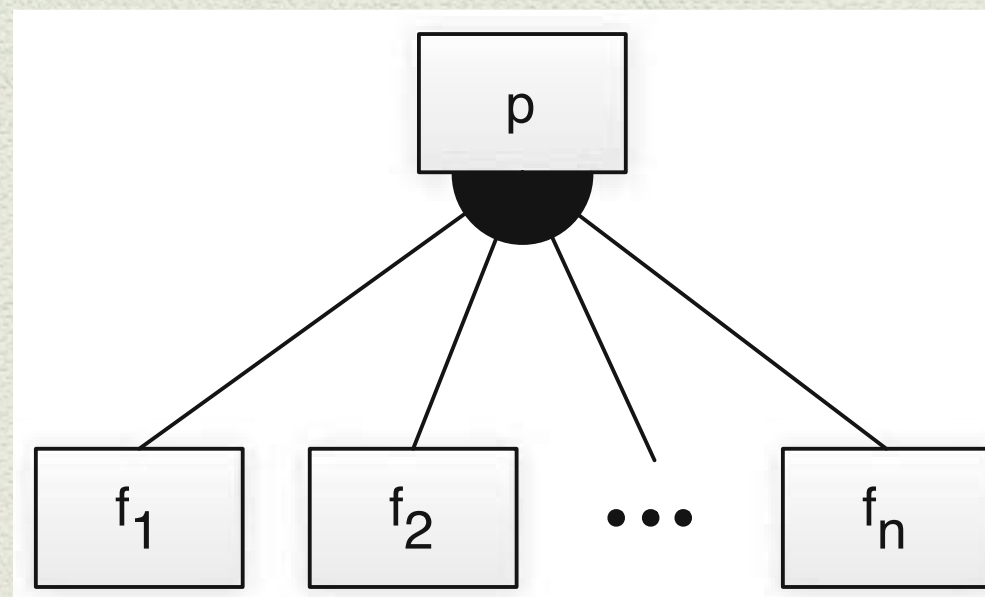
Generalises
previous slide



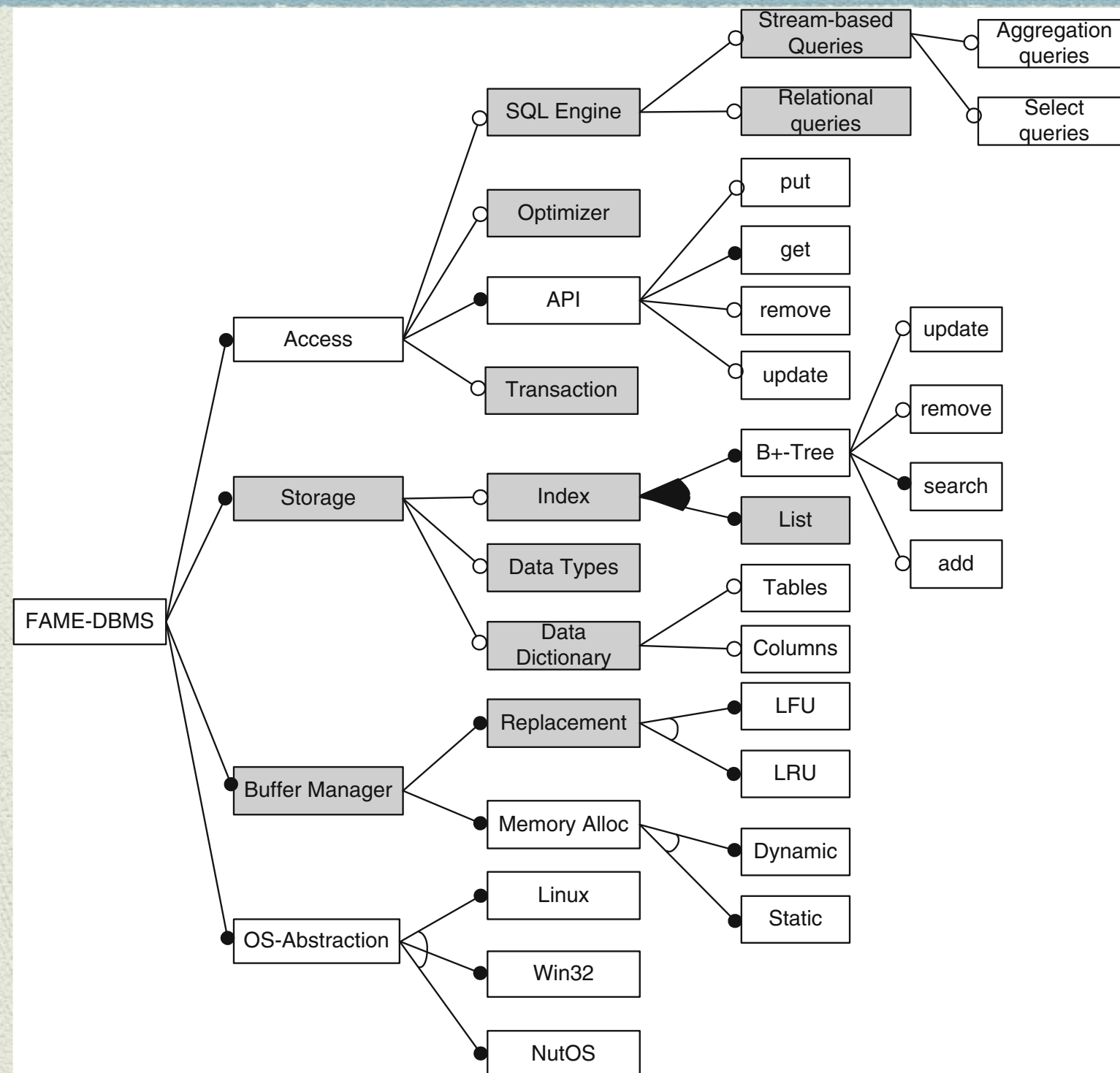
Alternative (XOR)



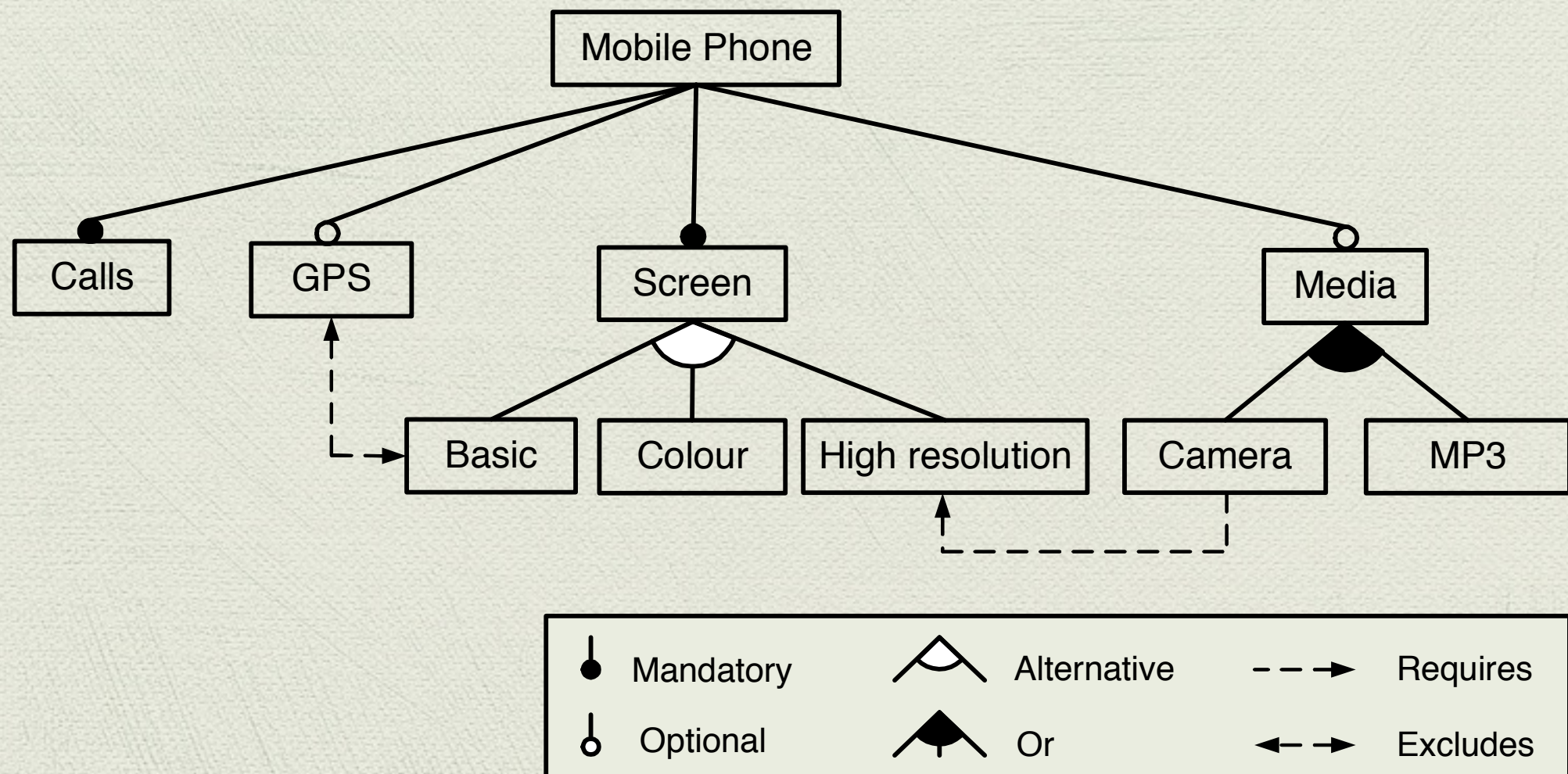
Some-Of-Many (OR)



Feature Model for Embedded Data Management



Cross tree constraints



Semantics

$$\text{root}(f) \equiv f$$

$$\text{mandatory}(p, f) \equiv f \Leftrightarrow p$$

$$\text{optional}(p, f) \equiv f \Rightarrow p$$

$$\begin{aligned} \text{alternative}(p, \{f_1, \dots, f_n\}) \equiv & ((f_1 \vee \dots \vee f_n) \Leftrightarrow p) \wedge \\ & \bigwedge_{i < j} \neg(f_i \wedge f_j) \end{aligned}$$

$$\text{or}(p, \{f_1, \dots, f_n\}) \equiv (f_1 \vee \dots \vee f_n) \Leftrightarrow p$$

What are the semantics of “implies” and “mutual exclusion”?

Analysis of Feature Models

- ◆ Void feature model
- ◆ Valid product
- ◆ Valid partial selection
- ◆ All products
- ◆ Number of products
- ◆ Filter (= all products for partial selection)
- ◆ Dead features
- ◆ False optional features
- ◆ Redundancies
- ◆ Obligatory features

Exercise: Develop Feature Model

- ◆ Identify features and interactions, conflicts and dependencies between features
- ◆ Design a feature model to capture the variability
 - ◆ Use abstract features to structure model better
 - ◆ Try to minimise cross-tree constraints (excludes and requires)

Return

- ◆ Experiences?
- ◆ What difficulties were experienced?

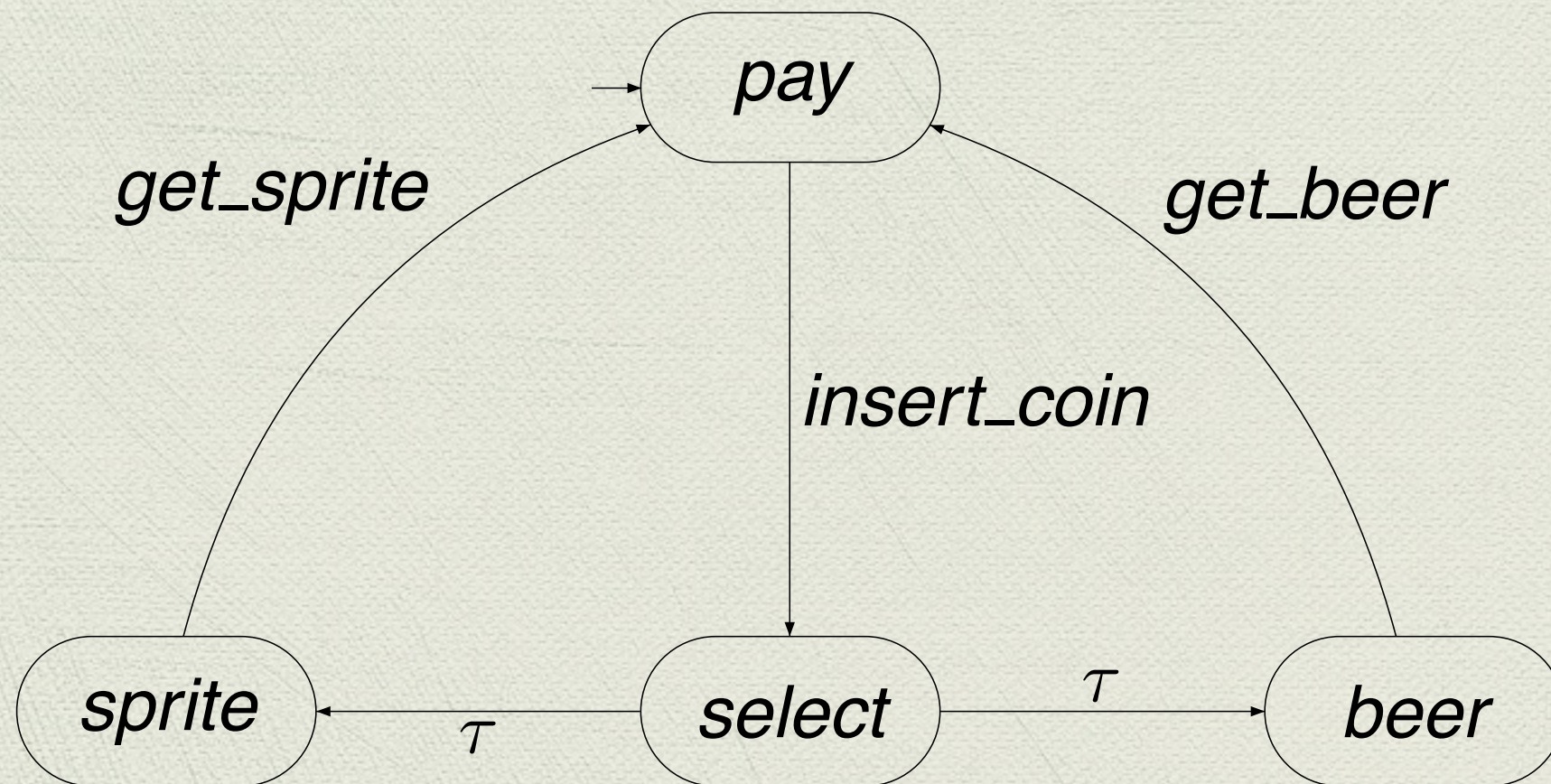
Behavioural Models

Behavioural Modelling

- ◆ Capture the behaviour of the entire software product line in a model
- ◆ Verify properties of model
 - ◆ Absolute
 - ◆ Feature-dependent
 - ◆ Which products satisfy property P ?

Labelled Transition Systems

Example: Vending Machine



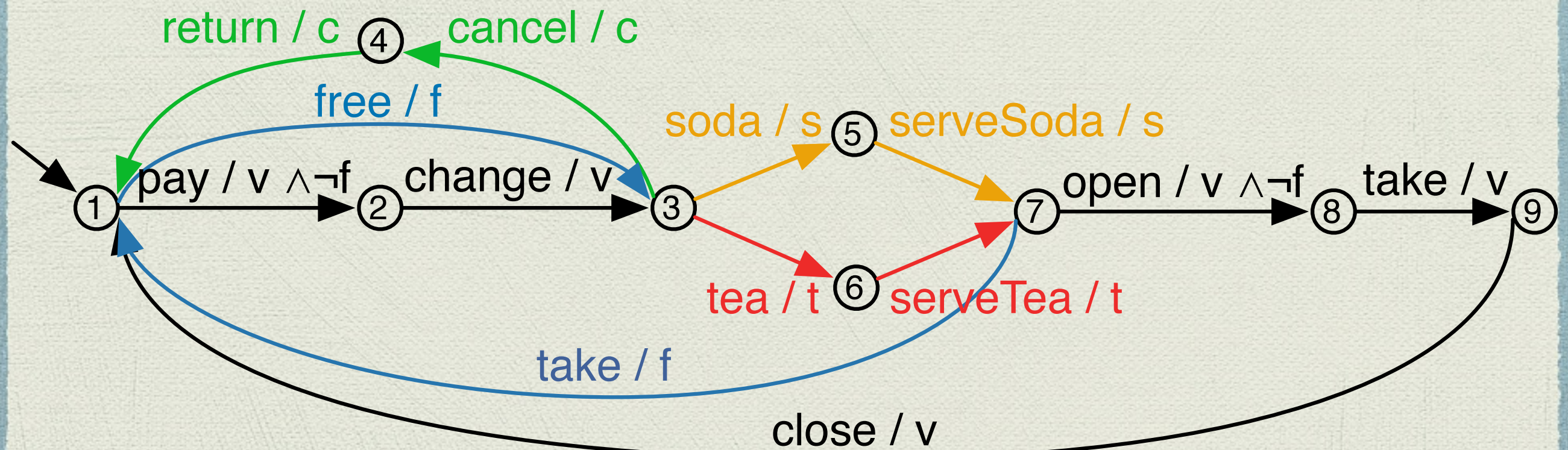
Labelled Transition Systems

A labelled transition system is a 4-tuple (S, Act, \rightarrow, I) , where

- S is a set of states
- Act is a set of actions
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation, and
- $I \subseteq S$ is a set of initial states.

$(s, a, s') \in \rightarrow$ is written $s \xrightarrow{a} s'$

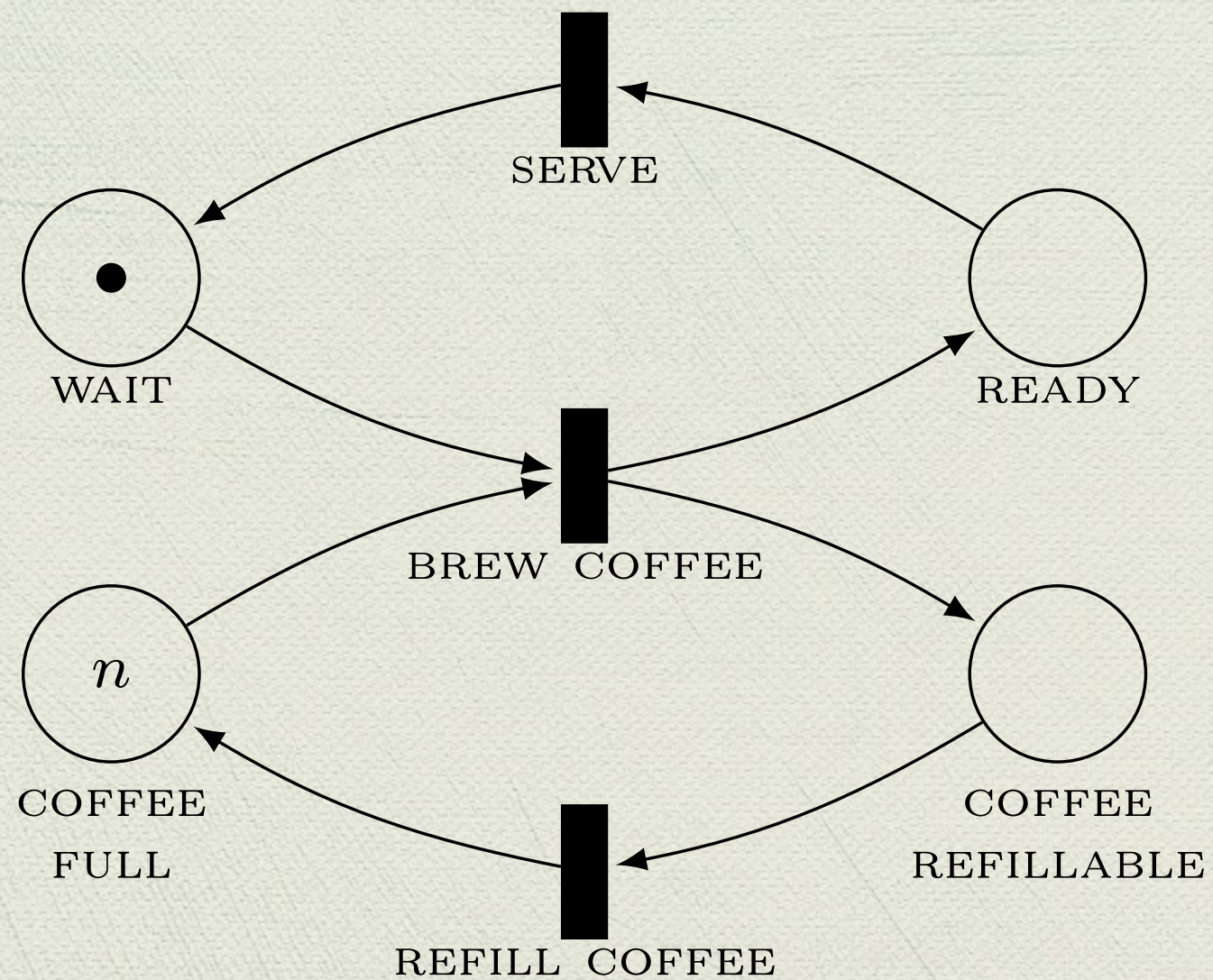
Featured Transition Systems



Critique

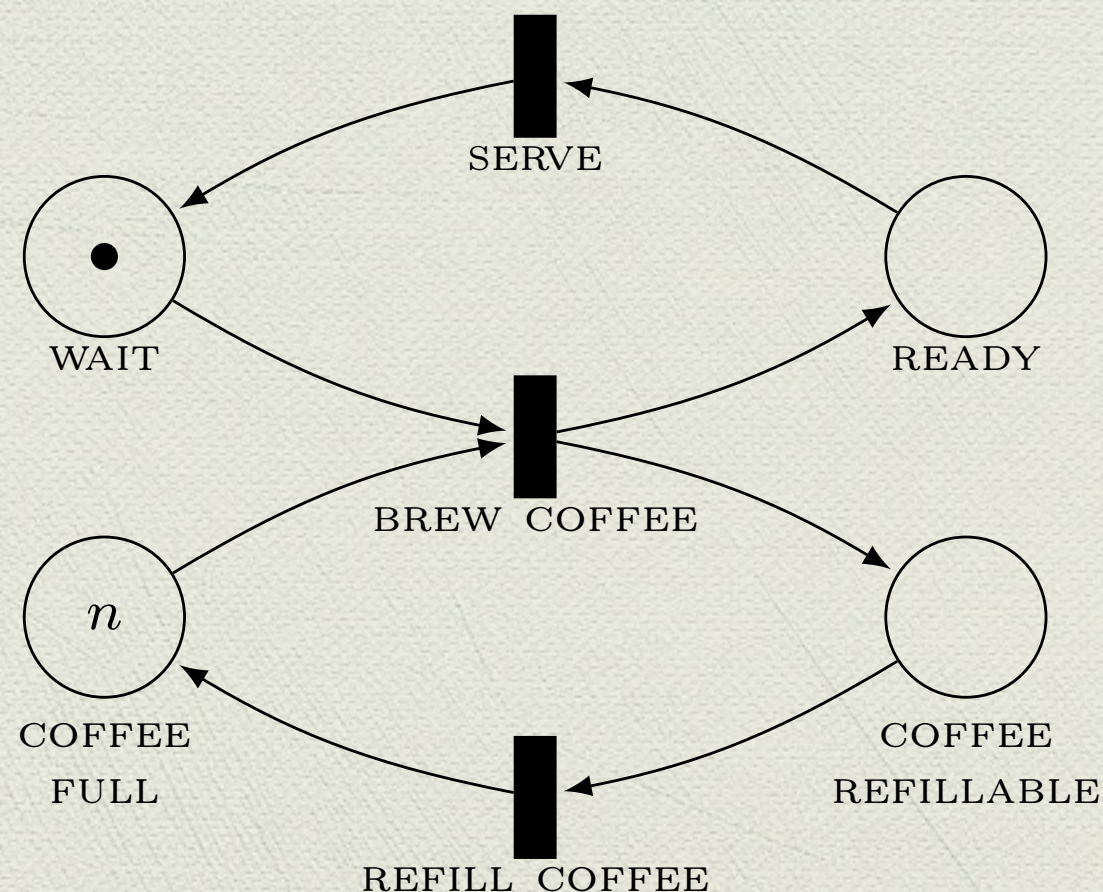
- ◆ LTL model checking tools have successfully been applied to FTS
- ◆ Featured transition systems tend to be very low level

Petri Nets



Definition

Definition 2 (Petri Net) A Petri net is a tuple (S, T, R, M_0) , where S and T are two disjoint finite sets, R is a relation on $S \cup T$ (the *flow relation*) such that $R \cap (S \times S) = R \cap (T \times T) = \emptyset$, and M_0 is a multiset over S , called the *initial marking*. The elements of S are called *places* and the elements of T are called *transitions*. Places and transitions are called *nodes*.



Semantics: Enabling

Definition 4 (Pre-sets and post-sets) Given a node x of a Petri net, the set $\bullet x = \{y \mid (y, x) \in R\}$ is the *pre-set* of x and the set $x^\bullet = \{y \mid (x, y) \in R\}$ is the *post-set* of x .

Definition 5 (Enabling) A marking M *enables* a transition $t \in T$ if it marks every place in $\bullet t$, that is, if $M \geq \bullet t$.

Semantics: Transitions

Definition 6 (Transition occurrence rule) Given a Petri net $N = (S, T, R)$, a transition $t \in T$ occurs, leading from a state with marking M_i to a state with marking M_{i+1} , denoted $M_i \xrightarrow{t} M_{i+1}$, iff the following two conditions are met:

$$M_i \geq \bullet t \quad \text{(enabling)}$$

$$M_{i+1} = (M_i - \bullet t) + t^\bullet \quad \text{(computing)}$$

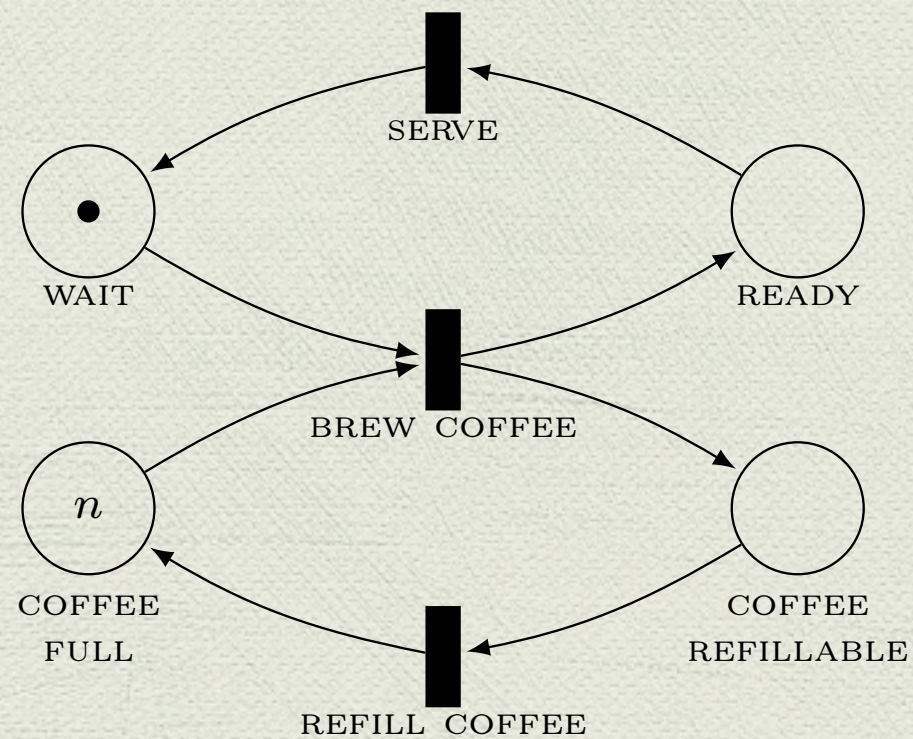
Semantics: Traces and Behaviour

Definition 7 (Petri net trace) Given a Petri net $N = (S, T, R, M_0)$, the behaviour the net exhibits by passing through a sequence of states with markings M_0, \dots, M_n , where each change of marking is triggered by a transition occurrence $M_i \xrightarrow{t_i} M_{i+1}$, is called a *trace*. A trace is written $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} M_n$.

Definition 8 (Petri net behaviour) The behaviour of a Petri net is given by the set of all traces from a given initial marking.

$$\text{Beh}(N) = \{M_0 \xrightarrow{t_1} \dots \xrightarrow{t_s} M_n \mid M_i \subseteq S, i \in 1..n, M_{i-1} \xrightarrow{t_i} M_i\}$$

Example



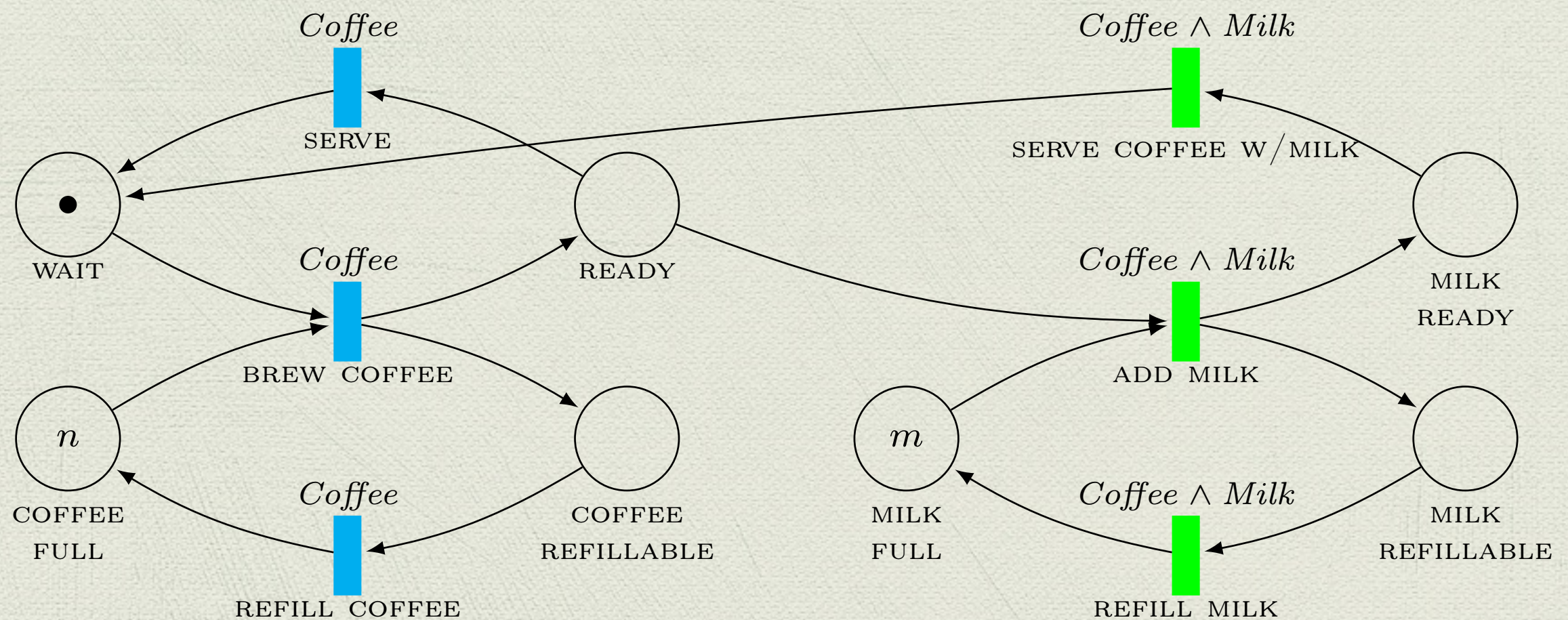
$$\begin{array}{l}
 \text{WAIT} \\
 \text{READY} \\
 \text{COFFEE FULL} \\
 \text{COFFEE REFILLABLE}
 \end{array}
 \begin{pmatrix} 1 \\ 0 \\ n \\ 0 \end{pmatrix}
 \xrightarrow{\text{BREW COFFEE}}
 \begin{pmatrix} 0 \\ 1 \\ n-1 \\ 1 \end{pmatrix}
 \xrightarrow{\text{SERVE}}
 \begin{pmatrix} 1 \\ 0 \\ n-1 \\ 1 \end{pmatrix}$$

Convert Petri net into LTS.

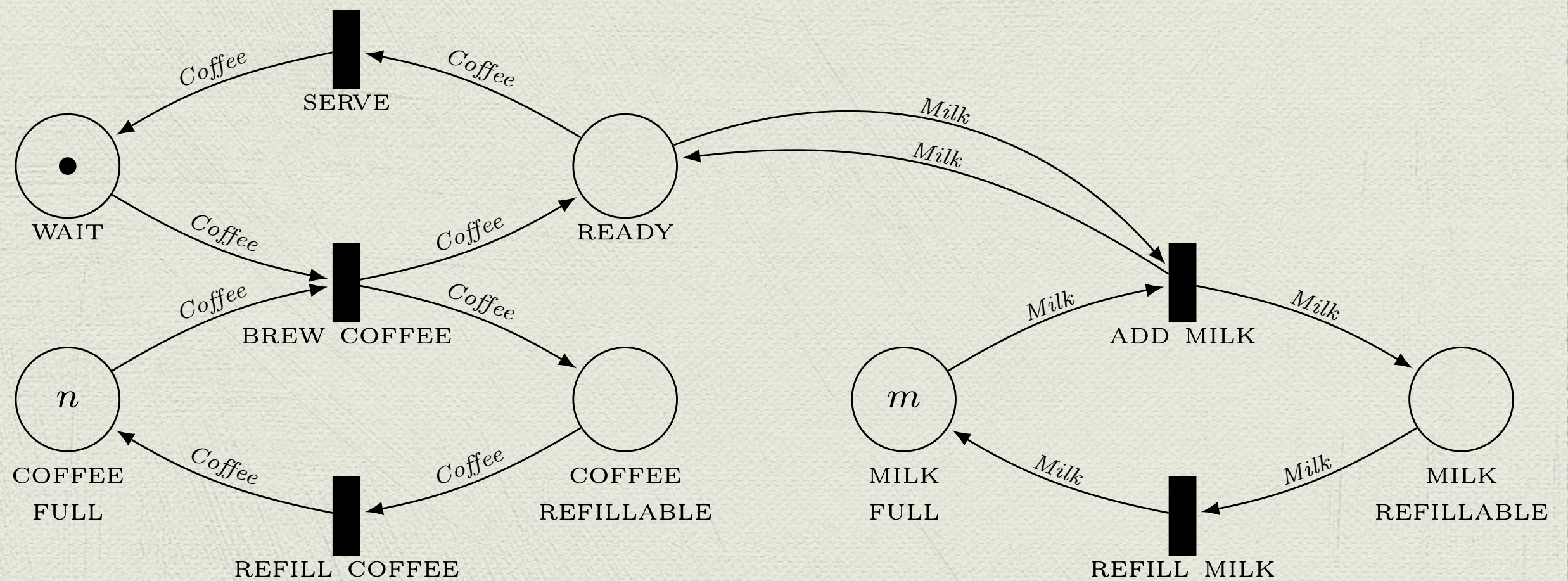
Feature Petri nets

- ◆ A Petri net extension to describe software product lines.
- ◆ Extends Petri nets in the same way that FTS extends LTS
- ◆ Two variants: application conditions on transitions or on arcs

Transition-Labelled Feature Petri Nets



Arc-Labelled Feature Petri Net



Semantics

Application Conditions

Definition 2 (Application condition [21]). An application condition φ is a propositional formula over a set of features F , defined by the following grammar:

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \neg \varphi,$$

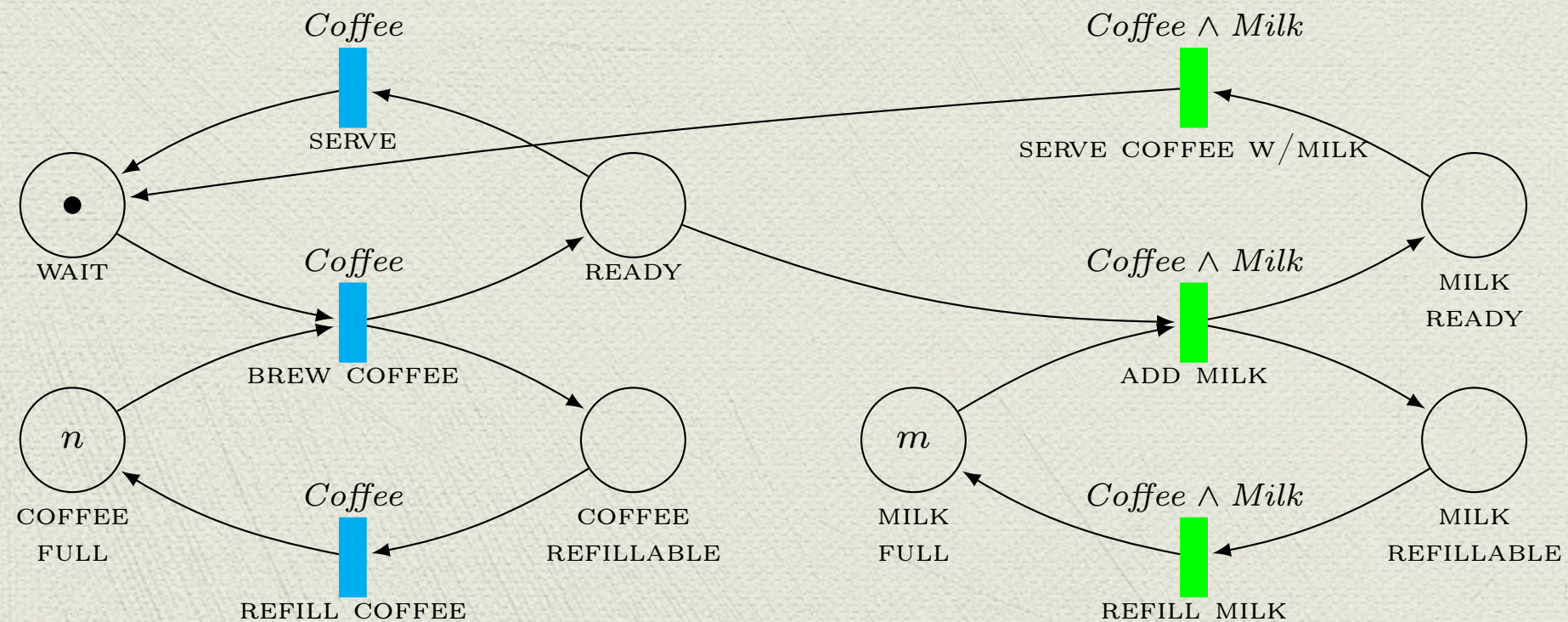
where $a \in F$. Write Φ_F to denote the set of all application conditions over F .

Definition 3 (Satisfaction of application conditions). Given an application condition $\varphi \in \Phi_F$ and a set of features $FS \subseteq F$, called a feature selection, we say that FS satisfies φ , written as $FS \models \varphi$, defined as follows:

$FS \models a$	iff $a \in FS$
$FS \models \varphi_1 \wedge \varphi_2$	iff $FS \models \varphi_1$ and $FS \models \varphi_2$
$FS \models \neg \varphi$	iff $FS \not\models \varphi$.

Semantics: Feature Nets

Definition 12 (Feature Net) A Feature Net is a tuple $N = (S, T, R, M_0, F, f)$, where (S, T, R, M_0) is a Petri net, F is set of features, and $f : T \rightarrow \Phi_F$ is a function associating each transition with an application condition from Φ_F .



Semantics: Transition Occurrence

Definition 13 (Transition occurrence rule for FN) Given a Feature Net $N = (S, T, R, M_0, F, f)$ and a feature selection $FS \subseteq F$, a transition $t \in T$ occurs, leading from a state with marking M_i to a state with marking M_{i+1} , denoted $(M_i, FS) \xrightarrow{t} (M_{i+1}, FS)$, iff the following three conditions are met:

$$M_i \geq \bullet t \quad \text{(enabling)}$$

$$M_{i+1} = (M_i - \bullet t) + t^\bullet \quad \text{(computing)}$$

$$FS \models \varphi_t \quad \text{(satisfaction)}$$

Semantics

Definition 14 (FN Trace) Given a Feature Net $N = (S, T, R, M_0, F, f)$ and a feature selection $FS \subseteq F$, the behaviour the net exhibits by passing through a sequence of markings M_0, \dots, M_n , where each change of marking is triggered by a transition occurrence $(M_i, FS) \xrightarrow{t_i} (M_{i+1}, FS)$, is called a *trace over FS*. A trace is written $(M_0, FS) \xrightarrow{t_0} (M_1, FS) \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} (M_n, FS)$.

Semantics

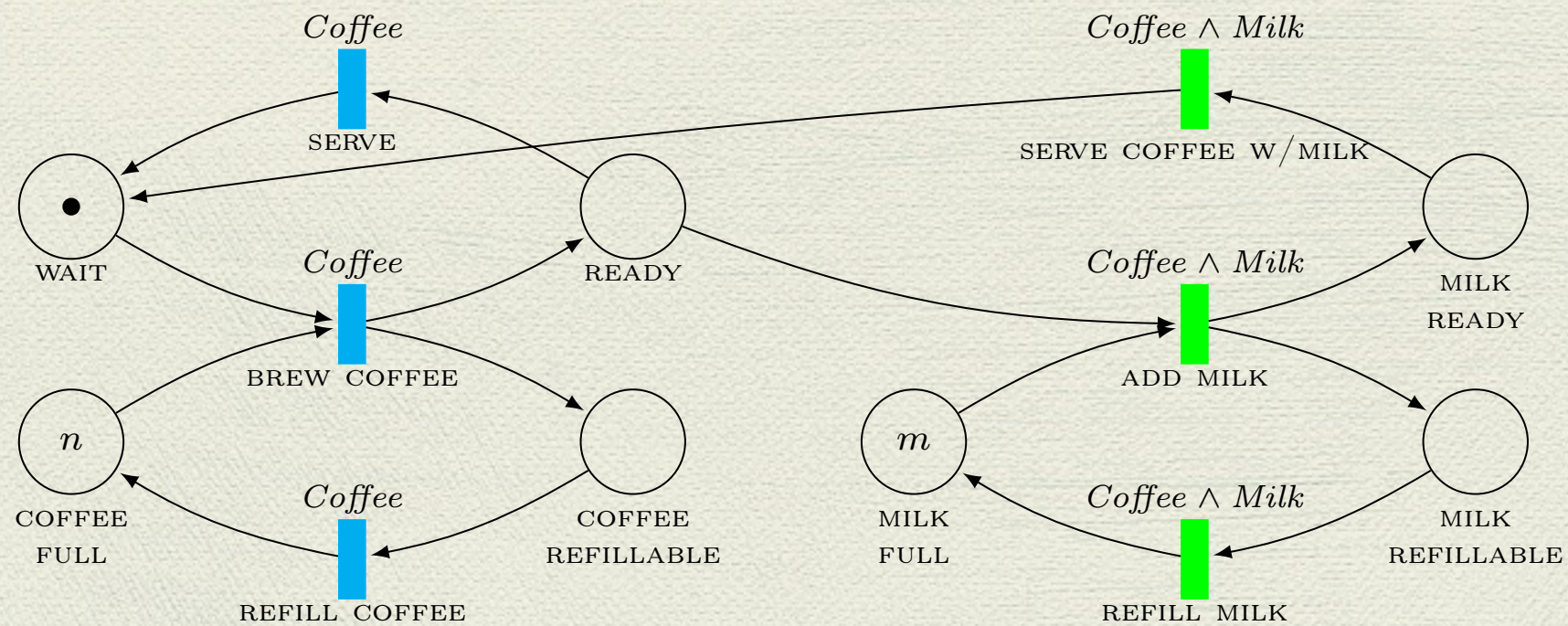
Definition 15 (FN behaviour for a given feature selection) Given a Feature Net $N = (S, T, R, M_0, F, f)$ and a feature selection $FS \subseteq F$, the behaviour of N for FS , denoted $\text{Beh}(N, FS)$ is the set of all traces over FS from the initial marking M_0 .

Definition 16 (FN Behaviour) Given a Feature Net $N = (S, T, R, M_0, F, f)$, we define $\text{Beh}(N)$ to be the combined set of behaviours for all feature selections over F :

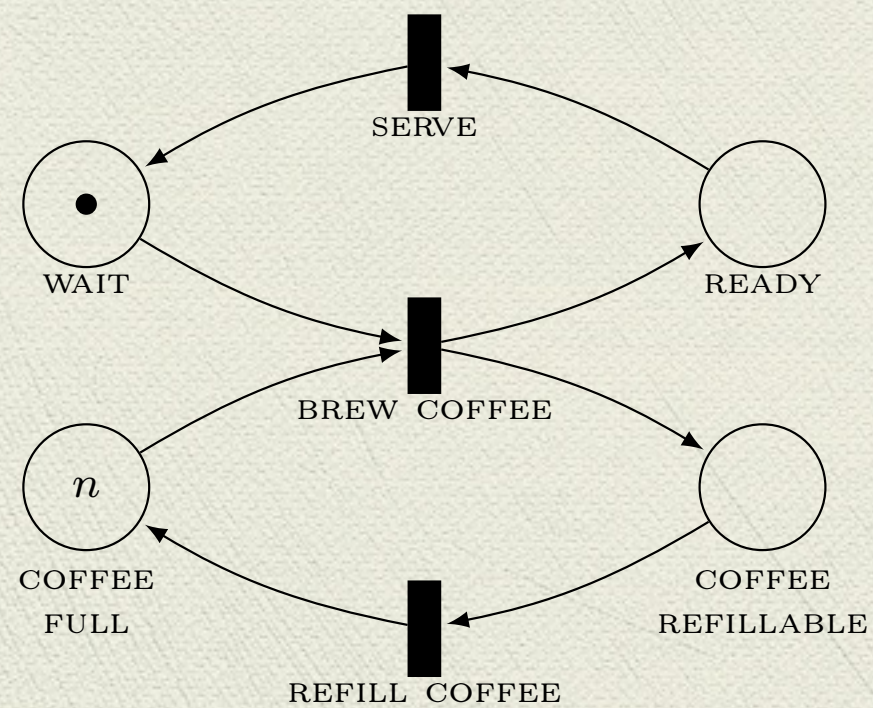
$$\text{Beh}(N) = \bigcup_{FS \in \mathcal{P}F} \text{Beh}(N, FS).$$

Projection-Based semantics

Definition 17 (Projection) Given a Feature Net $N = (S, T, R, M_0, F, f)$ and a feature selection $FS \subseteq F$, the *projection* of N onto FS , denoted $N \downarrow FS$, is a Petri net (S, T', R', M_0) , with $T' = \{t \in T \mid FS \models \varphi_t\}$ and the flow relation $R' = R \cap ((S \cup T') \times (S \cup T'))$.



$\downarrow \{Coffee\}$

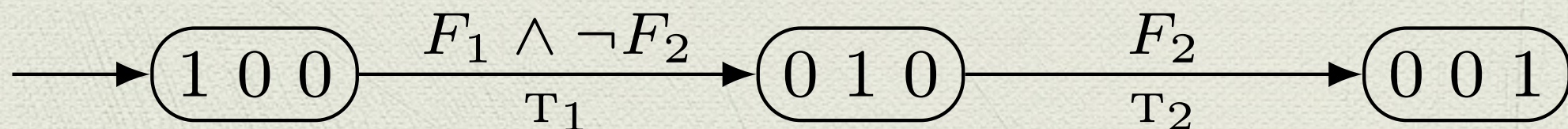
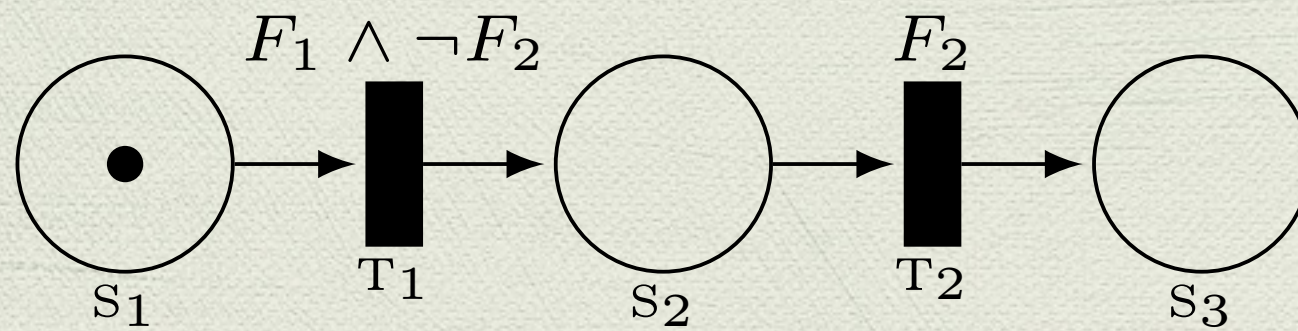


Equivalence of Semantics

Theorem 1 *Given a Feature Net N and $FS \subseteq F$, then:*

$$\text{Beh}(N, FS) \downarrow FS = \text{Beh}(N \downarrow FS).$$

Reachability



This is a feature transition system.
Can now use model checkers for those!

Exercise

- ◆ Apply Featured Transition Systems or Feature Petri nets to handed-out examples
- ◆ One strategy: model each feature independently, then combine.

Return

- ◆ Experiences?
- ◆ Pros and cons of each approach?

Delta Modelling

Running Example: Multi-lingual Hello World

- ◆ English: “Hello World”
- ◆ German: “Hallo Welt”
- ◆ Dutch: “Hallo Wereld”
- ◆ Swedish: “Hejsan Allihopa”
- ◆ French: “Bonjour tout le monde”
- ◆ Possibly with repetition

Ingredients of Variability in ABS

- ◆ Core ABS
- ◆ ~~Feature Model (μ TVL) — describing variability~~
- ◆ Deltas – implementing variability
- ◆ Configuration Language
- ◆ ~~Feature Selection Language~~

The Core

```
interface Greeting {  
    String say_hello();  
}
```

```
class Greeter implements Greeting {  
    String say_hello() {  
        return "Hello world";  
    }  
}
```

English by default



A fully functioning application
(minus main)

Delta Modelling

Delta-oriented Programming

- ◆ Modifications on Class Level:
 - ◆ Addition, Removal and Modification of Classes
- ◆ Modifications of Class Structure:
 - ◆ Changing interfaces
 - ◆ Adding / Removing Fields / Methods
 - ◆ Modifying Methods (wrapping original call)

The Repeat Delta

```
delta Rpt (Int times) {  
  modifies Greeter {  
    modifies String say_hello() {  
      String result = "";  
      Int i = 0;  
      while (i < times) {  
        result = result + original();  
        i = i + 1;  
      }  
      return result;  
    }  
  }  
}
```


The German Delta

```
delta De {  
    modifies Greeter {  
        modifies String say_hello() {  
            return "Hallo Welt";  
        }  
    }  
}
```


The Dutch Delta

```
delta N1 {  
    modifies Greeter {  
        modifies String say_hello() {  
            return "Hallo wereld";  
        }  
    }  
}
```


Product Line Configuration

Configuration

```
product line HelloMultiLingual {  
  features Repeat, German, French, Dutch, Swedish;  
  core English;  
  
  delta De when German && not Repeat;  
  delta Fr when French;  
  delta Nl when Dutch;  
  delta Sv when Swedish && Repeat;  
  delta Rpt(Repeat.times)  
  after De, Fr, Nl, Sv when Repeat;  
}
```

application conditions

parameter passing

ordering

Feature Selection

Feature Selection

Feature Selection

Attribute Specification

```
// apply delta Fr and Repeat
product P3 (French, Repeat{times=10}) {
  Greeting bob;
  bob = new Greeter();
  String s = "";
  s = bob.say_hello();
}
```

} Initialisation
(aka main)

Product Generation

Given Feature Selection

```
// apply delta Fr and Repeat  
product P3 (French, Repeat{times=10})  
{  
    Greeting bob;  
    bob = new Greeter();  
    String s = "";  
    s = bob.say_hello();  
}
```


Apply Delta Fr

```
class Greeter implements Greeting {  
    String say_hello() {  
        return "Hello world";  
    }  
}
```

+

```
delta Fr {  
    modifies Greeter {  
        modifies String say_hello() {  
            return "Bonjour tout le monde";  
        }  
    }  
}
```


Apply Delta Fr

```
class Greeter implements Greeting {  
    String say_hello() {  
        return "Bonjour tout le monde";  
    }  
}
```


Configure Repeat

times=10



```
delta Rpt (Int times) {  
  modifies Greeter {  
    modifies String say_hello() {  
      String result = "";  
      Int i = 0;  
      while (i < times) {  
        result = result + original();  
        i = i + 1;  
      }  
      return result;  
    }  
  }  
}
```


Configure Repeat

```
delta Rpt {
  modifies Greeter {
    modifies String say_hello() {
      String result = "";
      Int i = 0;
      while (i < 10) {
        result = result + original();
        i = i + 1;
      }
      return result;
    }
  }
}
```


Apply Repeat

```
class Greeter implements Greeting {  
    String say_hello() {  
        return "Bonjour tout le monde";  
    }  
}
```

+

```
delta Rpt {  
    modifies Greeter {  
        modifies String say_hello() {  
            String result = "";  
            Int i = 0;  
            while (i < 10) {  
                result = result + original();  
                i = i + 1;  
            }  
            return result;  
        }  
    }  
}
```


Apply Repeat


```
class Greeter implements Greeting {  
    String __say_hello_original() {  
        return "Bonjour tout le monde";  
    }  
  
    String say_hello() {  
        String result = "";  
        Int i = 0;  
        while (i < 10) {  
            result = result + __say_hello_original();  
            i = i + 1;  
        }  
        return result;  
    }  
}
```


Adding Initialisation

```
class Greeter implements Greeting {
    String __say_hello_original() {
        return "Bonjour tout le monde";
    }

    String say_hello() {
        String result = "";
        Int i = 0;
        while (i < 10) {
            result = result + __say_hello_original();
            i = i + 1;
        }
        return result;
    }
}

{
    Greeting bob;
    bob = new Greeter();
    String s = "";
    s = bob.say_hello();
}
```



Another Example

Delta Modelling Example

```
interface IAccount { Unit deposit(Int x); }

class Account implements IAccount { // Core Product
    Int balance = 0;
    Unit deposit(Int x) {
        balance = balance + x;
    }
}

delta DFee(Int fee) { // Implements Feature Fee
    modifies class Account {
        modifies Unit deposit(Int x) {
            if (x >= fee) original(x-fee);
        }
    }
}
```


Delta Modelling Example (cont'd)

```
delta DOverdraft() { // Implements Feature Overdraft
  modifies class Account {
    adds Int limit;
    modifies Unit deposit(Int x) {
      if (balance + x > limit) original(x);
    }
  }
}

productline AccountPL { // Product Line Declaration
  features Basic, Overdraft, Fee;
  delta DFee(Fee.amount) when Fee;
  delta DOverdraft after DFee when Overdraft;
}
```


Exercise

- ◆ Apply delta modelling or `#ifdefs` to handed out code.
- ◆ Currently code includes only two products from product line. Can you make it express all?

Return

- ◆ Experiences?
- ◆ Pros and cons of each approach?
- ◆ How would selected approach scale to larger project?

Concluding Remarks

References

- ◆ Feature Petri Nets. Radu Muschevici, Dave Clarke, José Proença. SPLC Workshops 2010: 99-106
- ◆ Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-Francois Raskin IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. 8, AUGUST 2013
- ◆ Delta-oriented Programming of Software Product Lines. Schaefer, Bettini, Bono, Damiani, Tanzarella. SPLC 2010.
- ◆ Variability Modelling in the ABS Language. Dave Clarke, Radu Muschevici, José Proença, Ina Schaefer, Rudolf Schlatte. FMCO 2010: 204-224