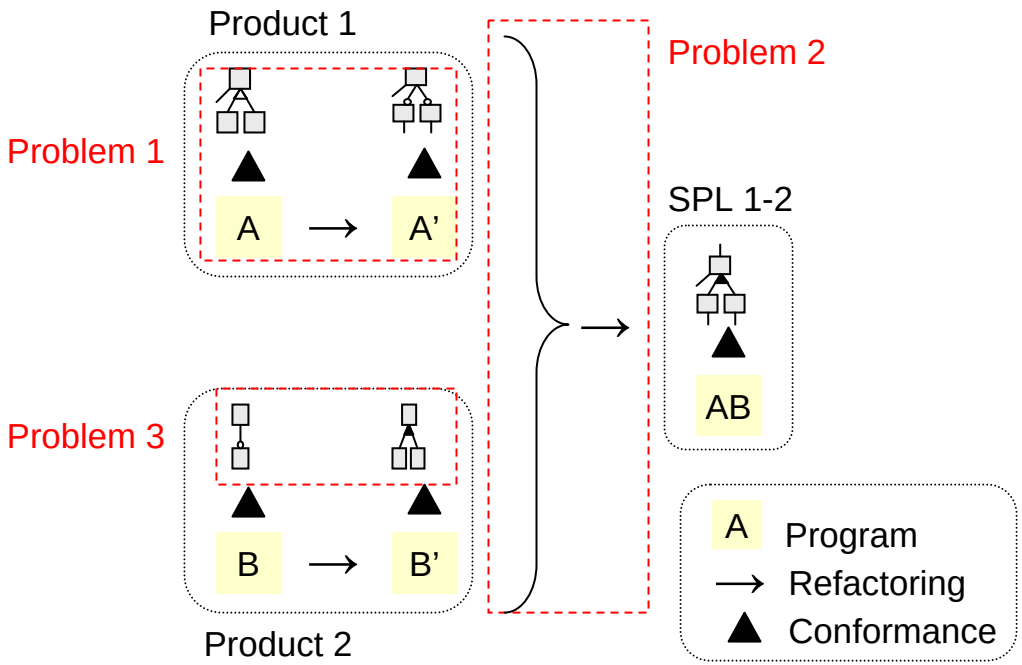
Refactoring Product Lines

Problem

Problems in Refactoring SPL



Feature Model Theory

Abstract Syntax

```
Name: TYPE
FM: TYPE = [#
    features: set[Name],
    formulae: set[Formula]
#]
Config: TYPE = [#
    config: set[Name]
#]
```

Formulae

- True
- False
- Name
- Not
- And
- Implies

Dynamic Semantics

```
semantics(fm: {f:FM | wf(f)}): set[Config] =
   { c: Config |
        satImpConst(fm,c) ∧
        satExpConst(fm,c)
   }
```

Constraints

```
satImpConst(fm:{f:FM | wf(f)}, c:Config): bool =
    config(c) ⊆ features(fm)
```

```
satExpConst(fm:{f:FM | wf(f)}, c:Config): bool =
    ∀ f: formulae(fm) | satFormula(f,c)
```

satFormula(f,c)

True, False : **true, false**

Name(n) : n ∈ config(c)

Not(f1) : **not** satFormula(f1,c)

And(f1,f2) : satFormula(f1,c) ∧
satFormula(f2,c)

Implies(f1,f2) : satFormula(f1,c) ⇒
satFormula(f2,c)

Static Semantics

```
wf(fm FM): bool =
    ∀ f: formulae(fm) | wellTyped(fm,f)
```

wellTyped(fm,f)

True, False : **true**

Name(n) : n ∈ features(fm)

Not(f1) : wellTyped(fm,f1)

And(f1,f2) : wellTyped(fm,f1) ∧
wellTyped(fm,f2)

Implies(f1,f2) : wellTyped(fm,f1) ∧
wellTyped(fm,f2)

Reactive PL Refinement

refines(abs, con:FM): bool =
  semantics(abs) ⊆ semantics(con)

Extractive PL Refinement

refines(abs1, abs2, con:FM): bool =

refines(abs1,con) ∧

refines(abs2,con)

Equivalences between Feature Models

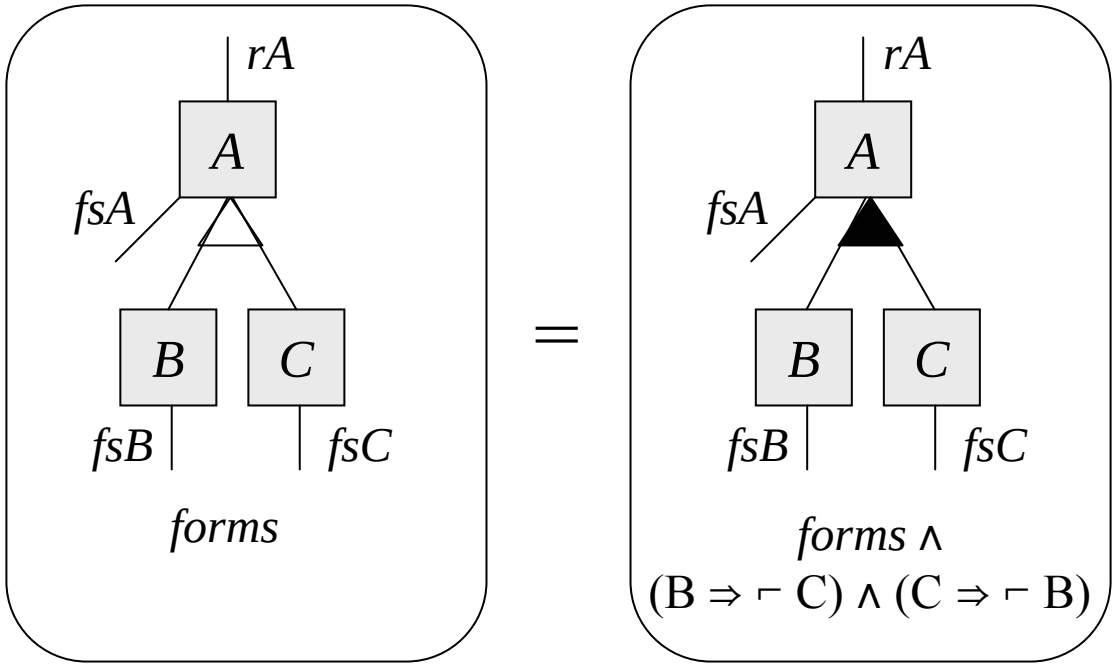
Core Language

- Our core language contains
  - features
  - formulae
- This core language is specified in PVS
- Notice that our core language does not have the same static semantics of the original one.
  - So, some of the syntactic sugar (Laws 4 and 5) will not be used in practice.
  - However, it is as expressive as the original one.

Syntactic Sugar Transformations

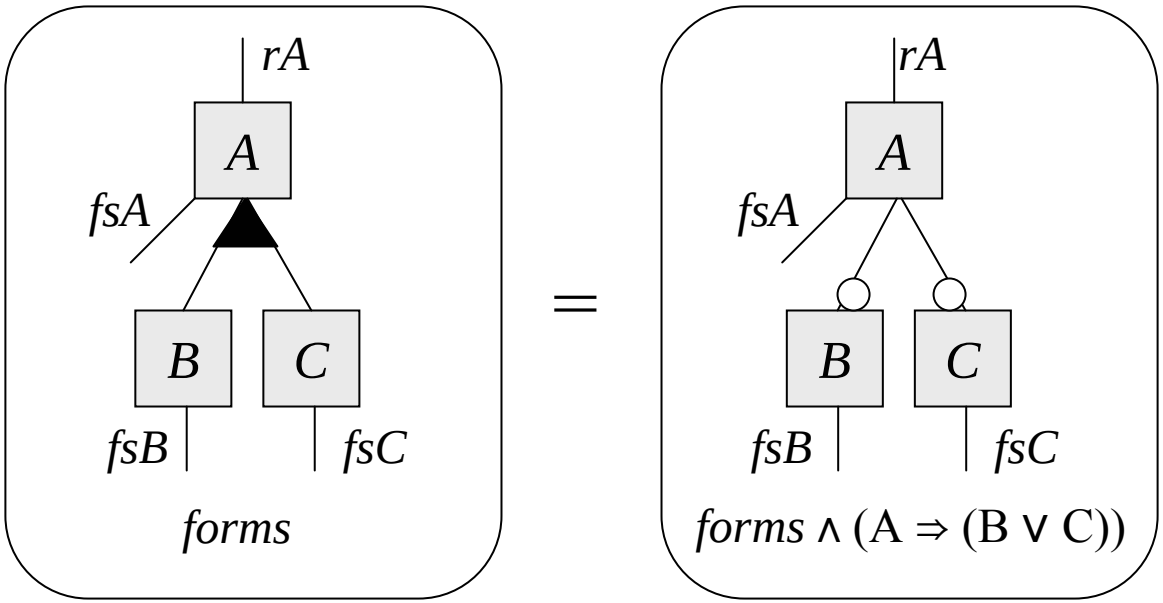
- They are intuitively valid by definition
- We do not need to prove them
- We use the syntactic sugar laws to reduce any feature model to our core language
- We propose 14 syntactic sugar refinements
- This set is sound, minimal and complete.

Replace Alternative (Law 1)

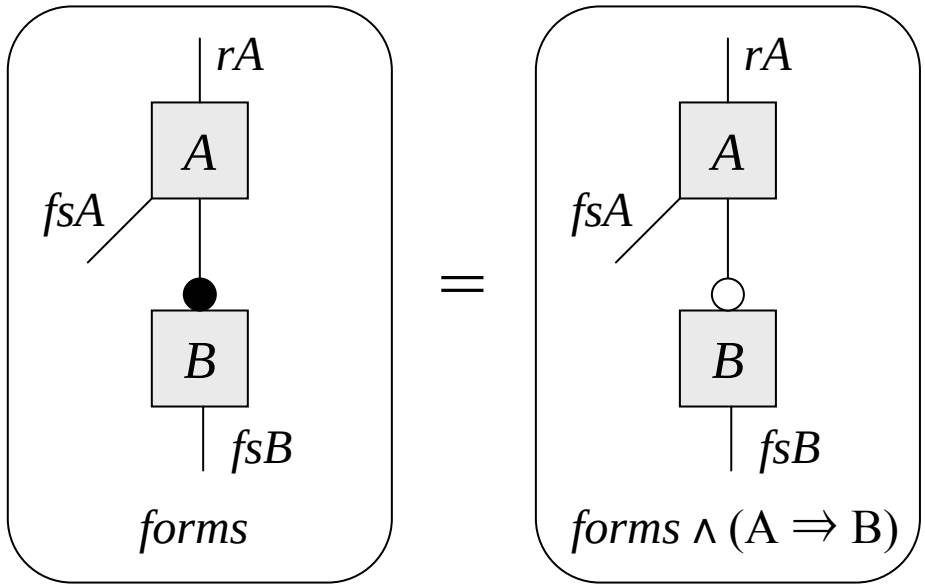


r: root
fs: features
forms: formulae

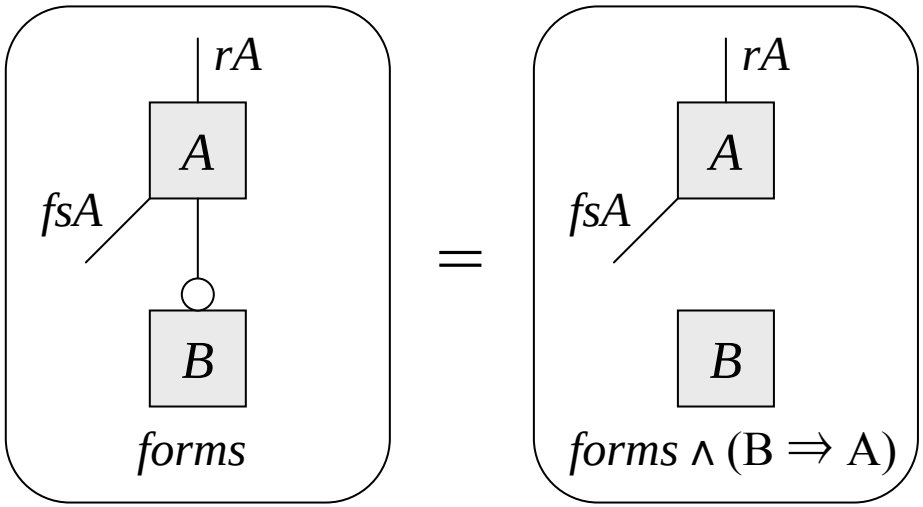
Replace Or (Law 2)



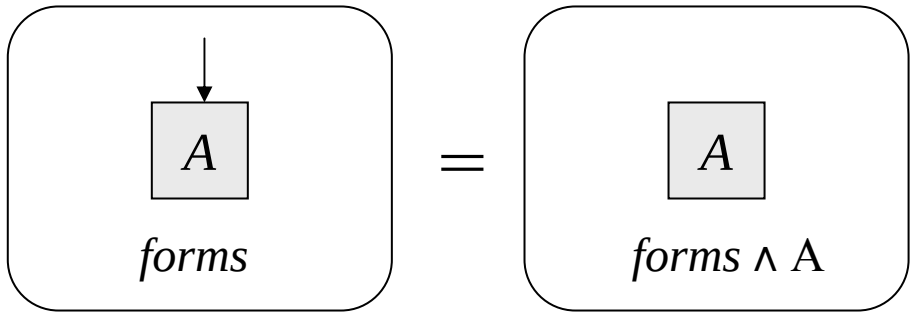
Replace Mandatory (Law 3)



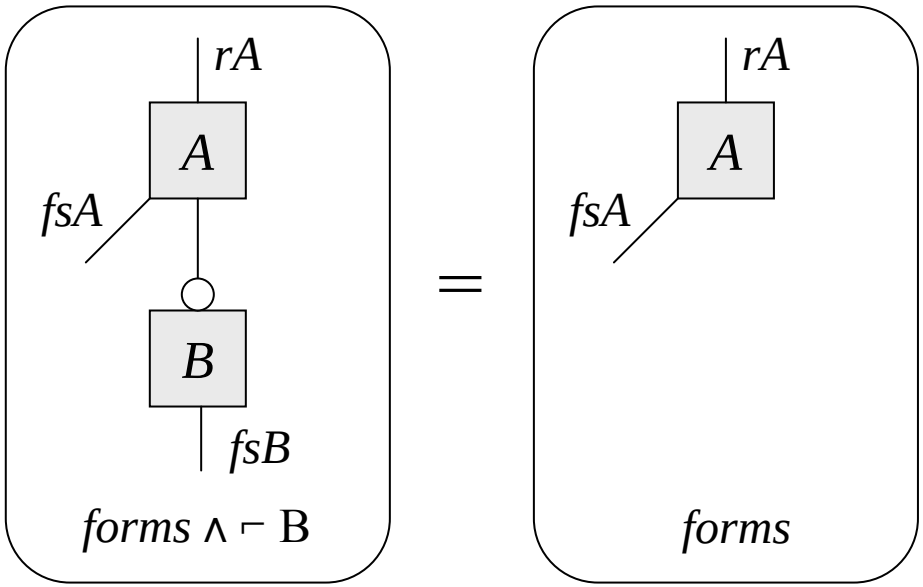
Replace Node (Law 4)



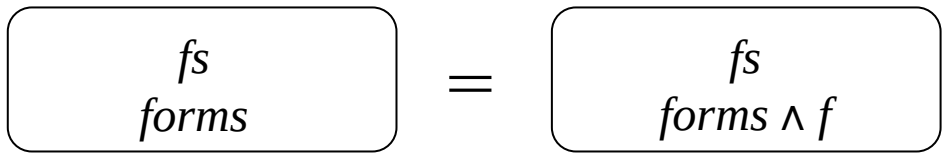
Remove Root (Law 5)



Remove Node (Law 6)



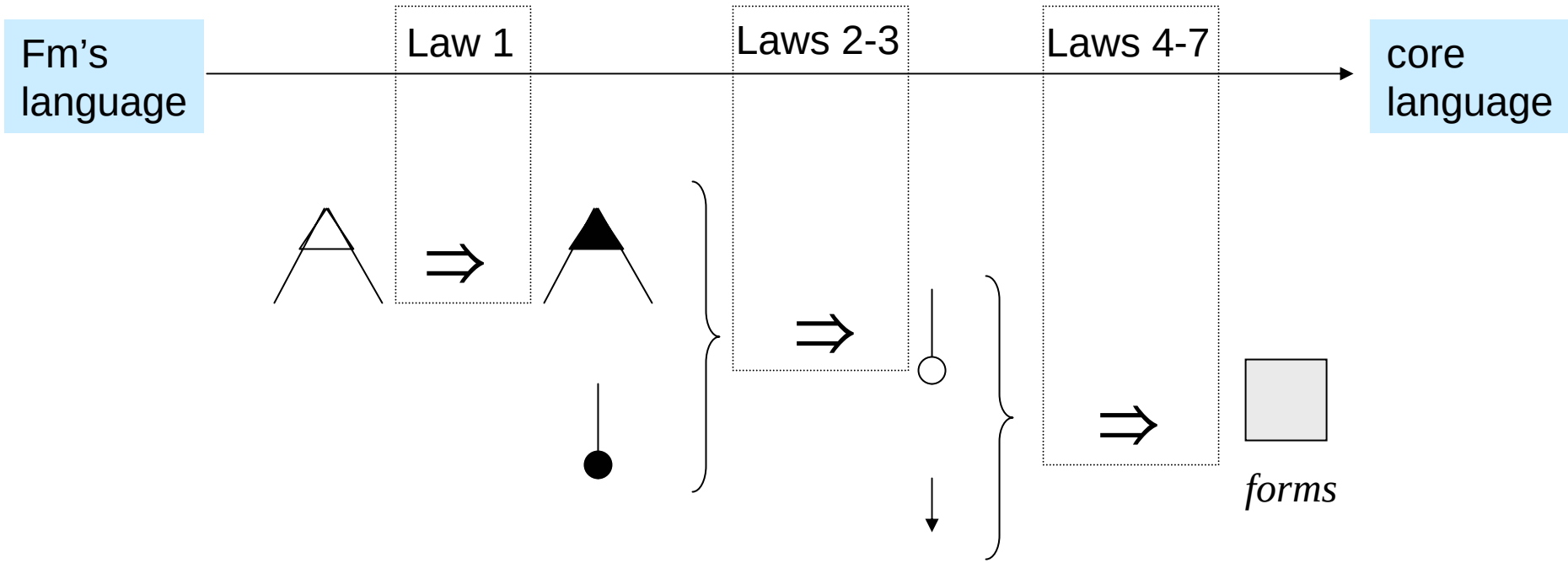
Add Formula (Law 7)



(↔) *f* can be deduced from *forms* and *fs*.

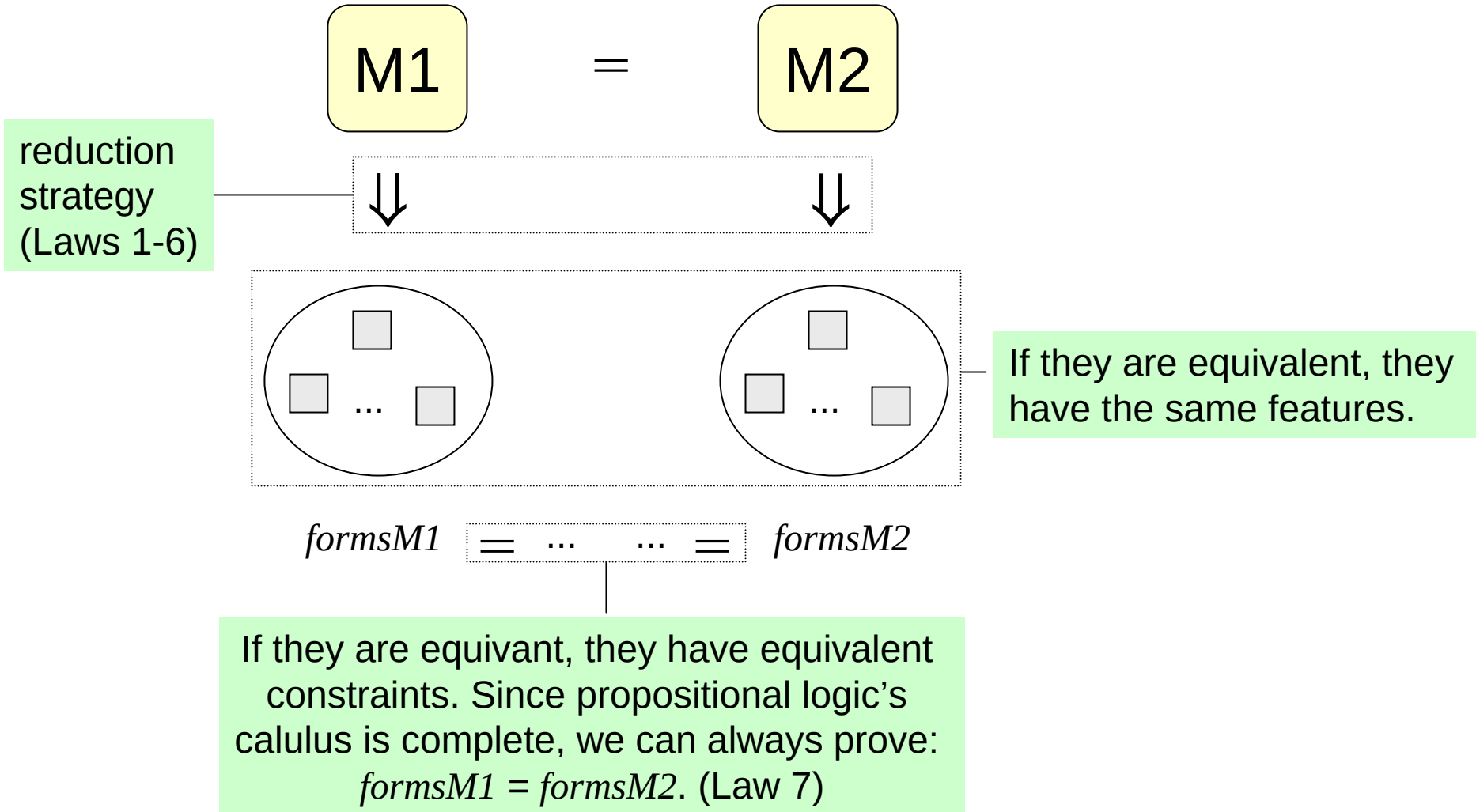
Proved in PVS

Reduction Strategy



All laws are applied from left to right

Completeness (proof)

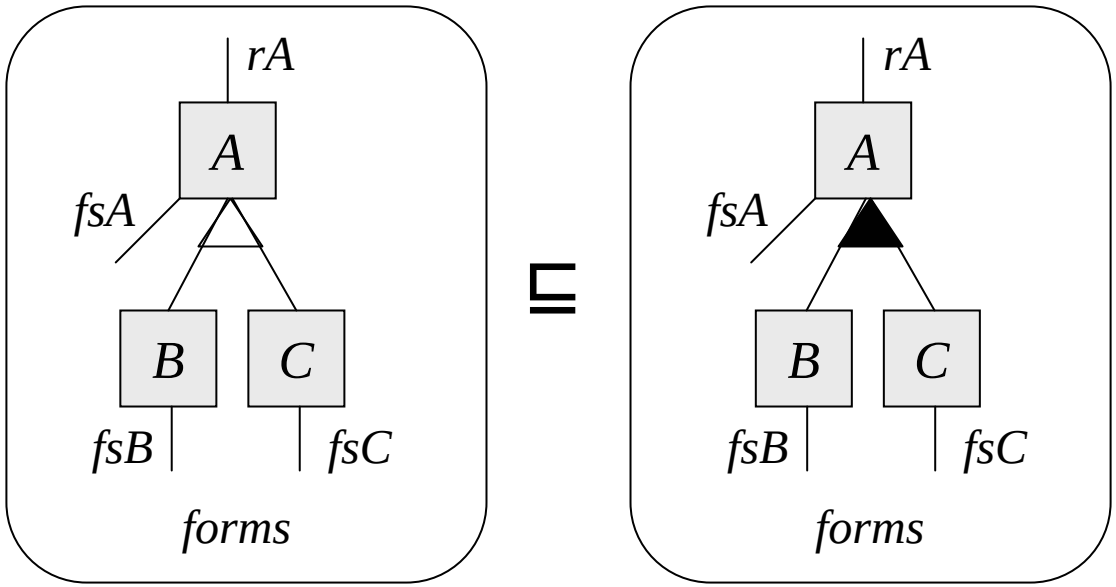


Refinements

Refinements

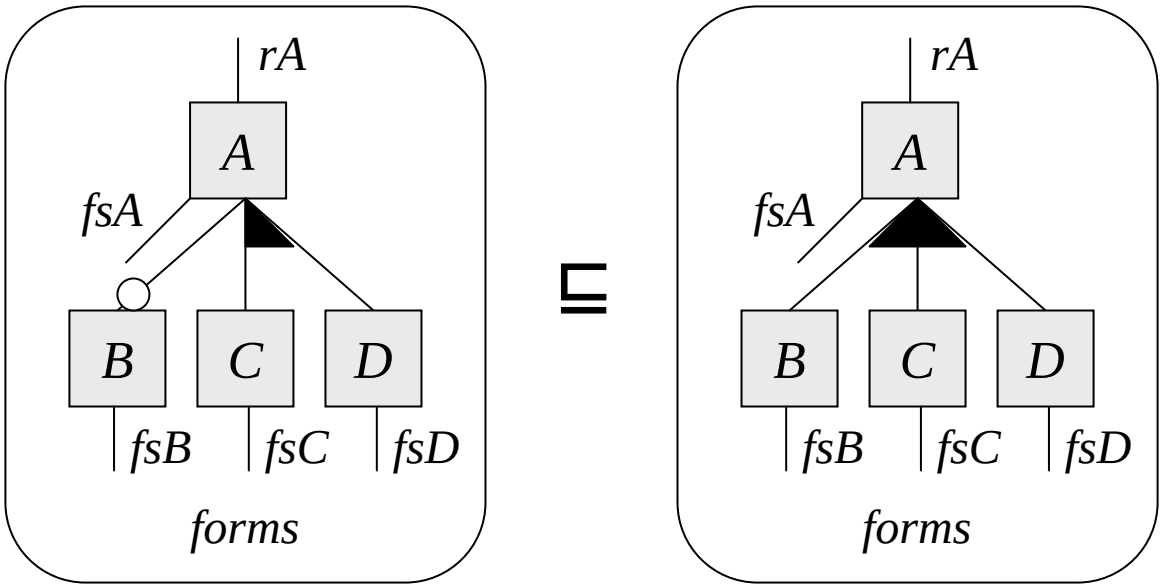
- We propose 12 refinements
- All of them are proved in PVS with respect to
  - static semantics
  - dynamic semantics
- It is important to mention that we have more transformations since most of them can be applied with more than two features

Change Alternative to Or (Ref 1)



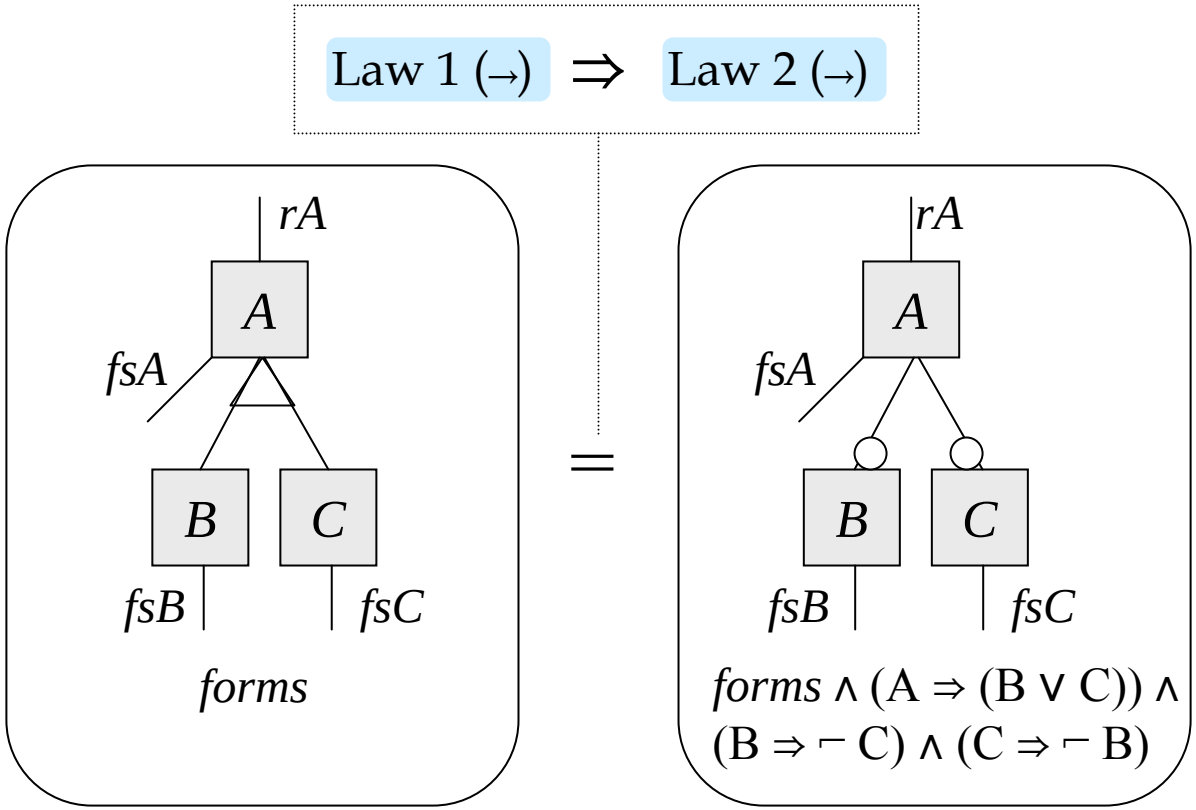
Proved in PVS

Collapse Optional and Or (Ref 2)

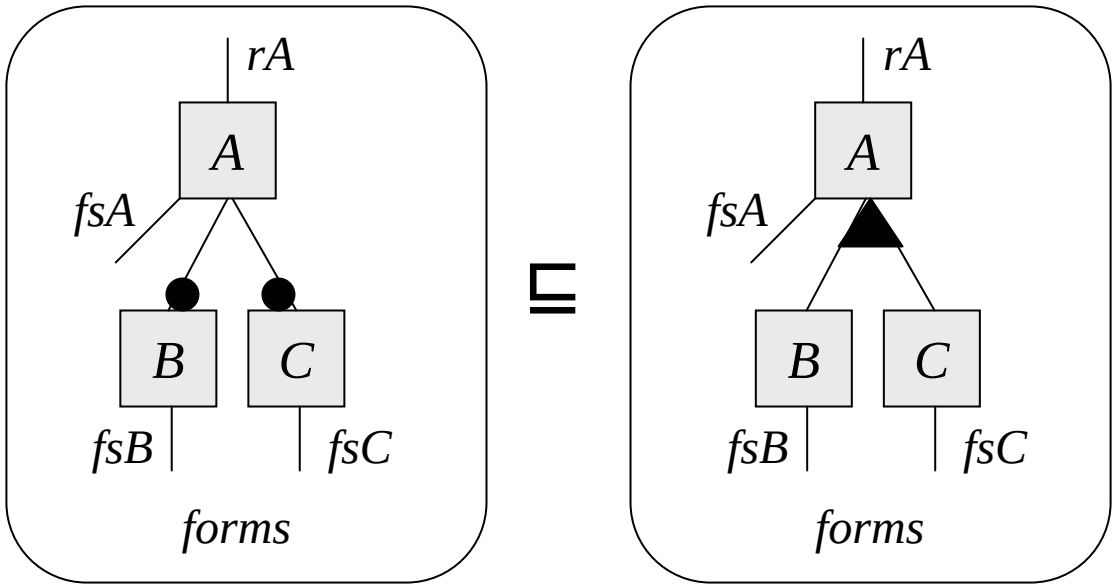


Proved in PVS

Change Alternative to Optional (Ref 3)

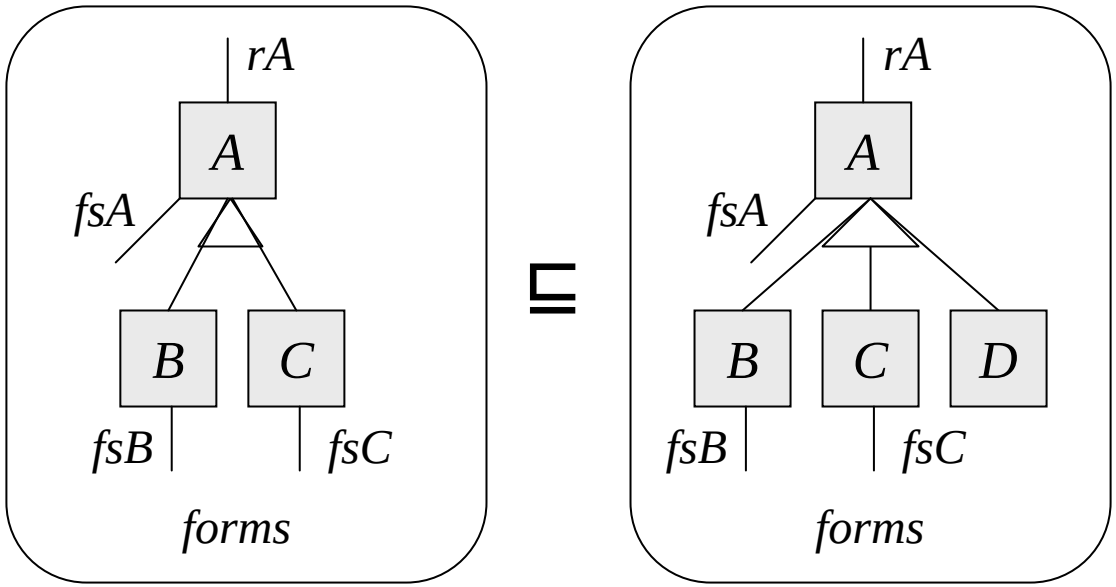


Add Or Between Mandatory (Ref 4)



Proved in PVS

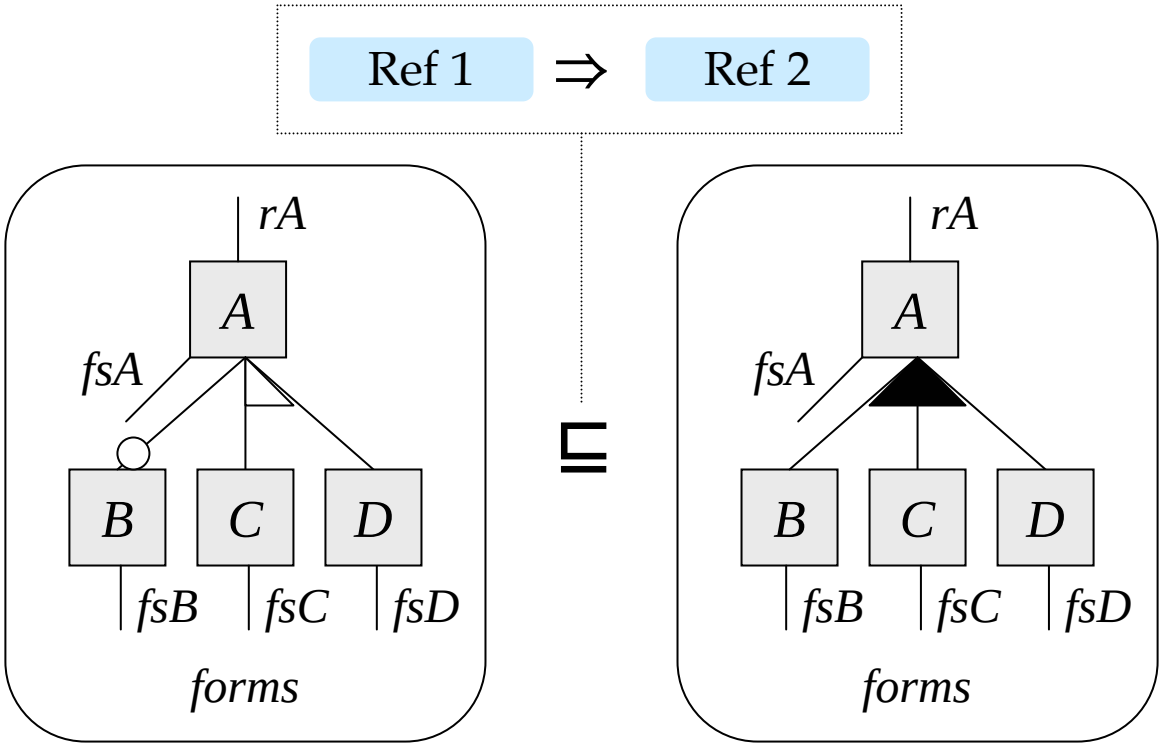
Add New Alternative (Ref 5)



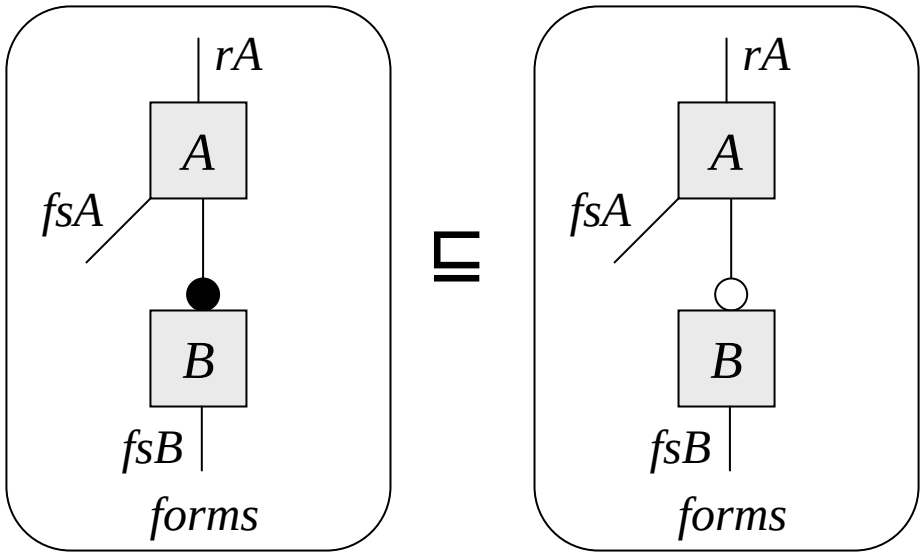
D is a new name

Proved in PVS

Collapse Optional and Alternative to Or (Ref 6)

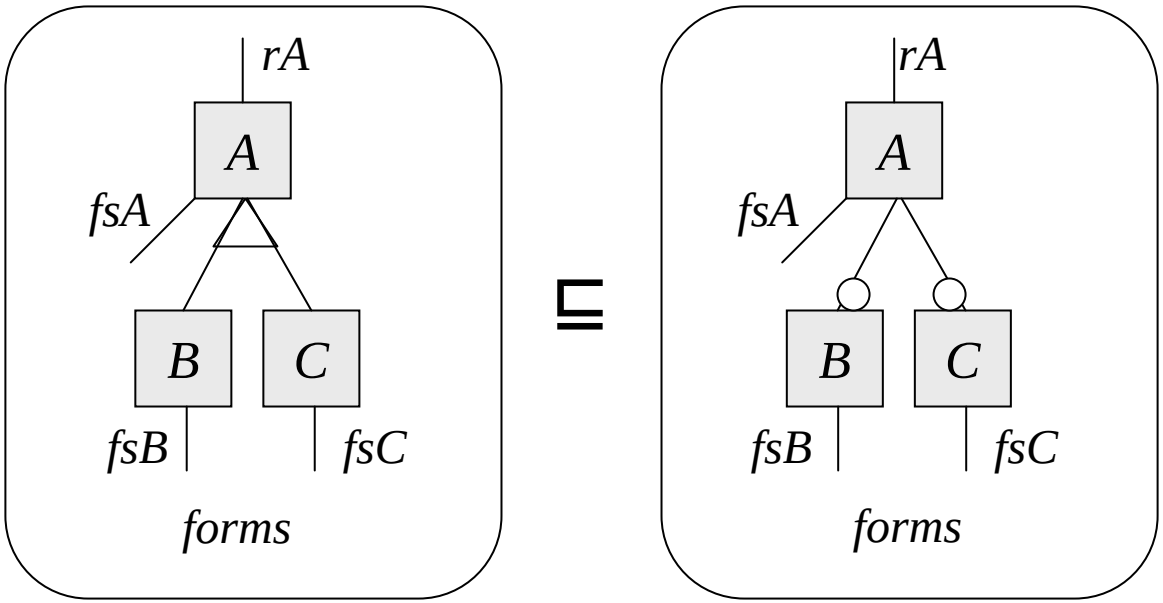


Remove Mandatory (Ref 7)



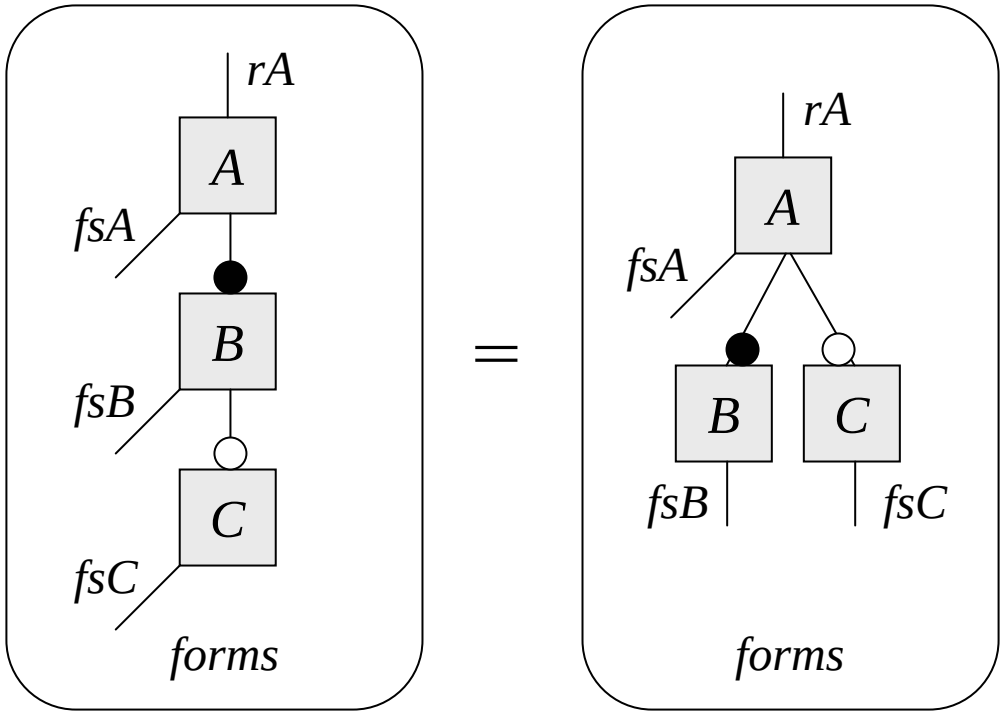
Proved in PVS

Remove Alternative (Ref 8)



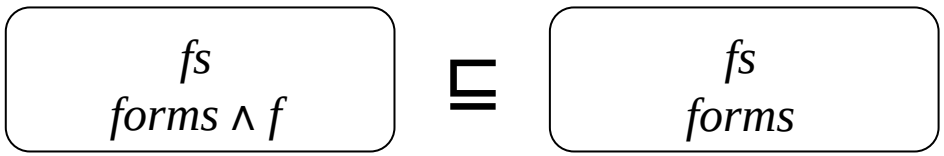
Proved in PVS

Pull up Node (Ref 9-10)



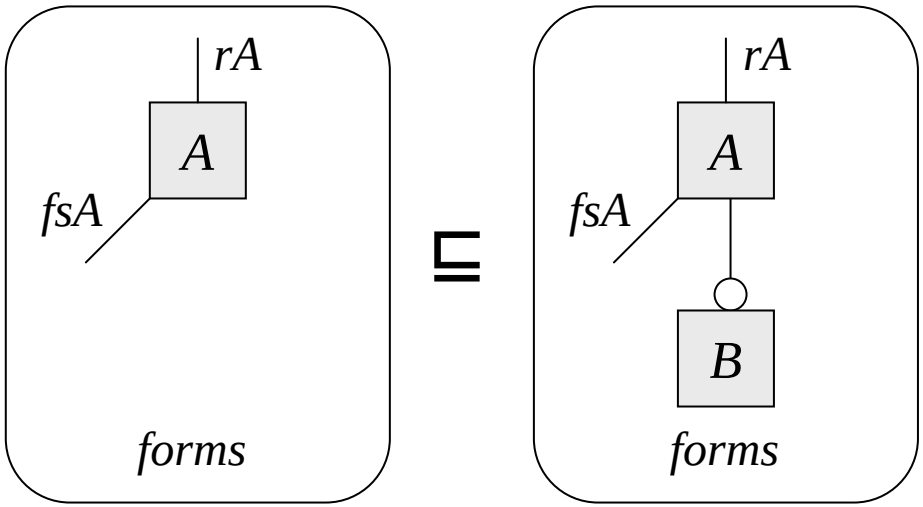
Proved in PVS

Remove Formula (Ref 11)



Proved in PVS

Add Optional Node (Ref 12)

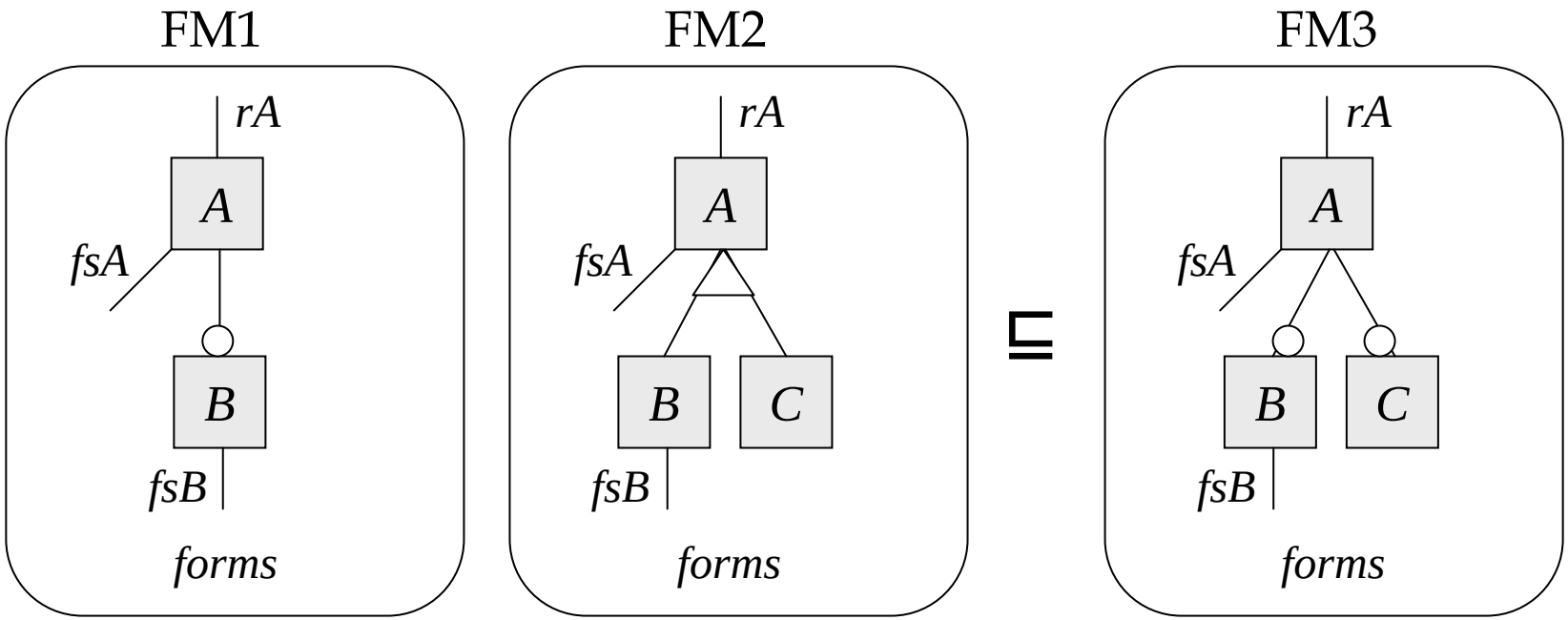


B is a new name

Proved in PVS

Extractive Refinements

Extractive 1

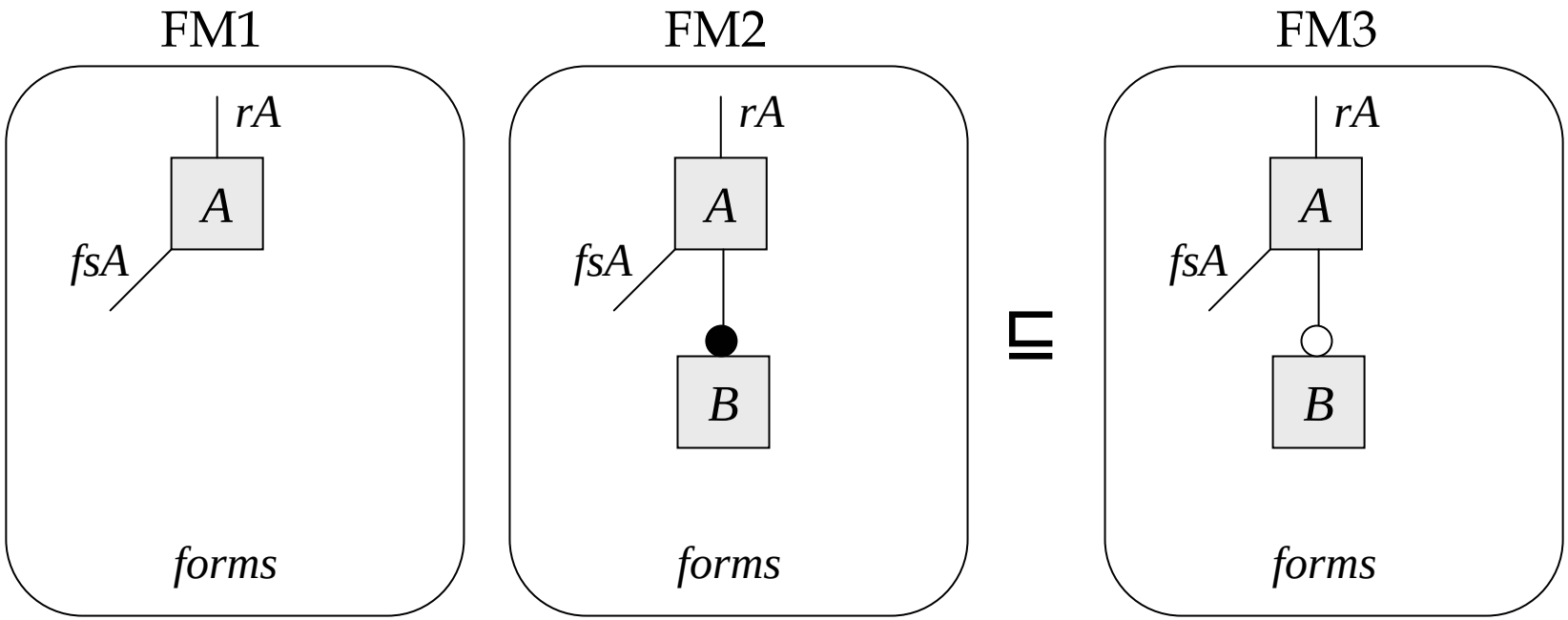


Proofs:

FM1 ⊑ FM3: Ref 12 (intro C)

FM2 ⊑ FM3: Ref 8

Extractive 2

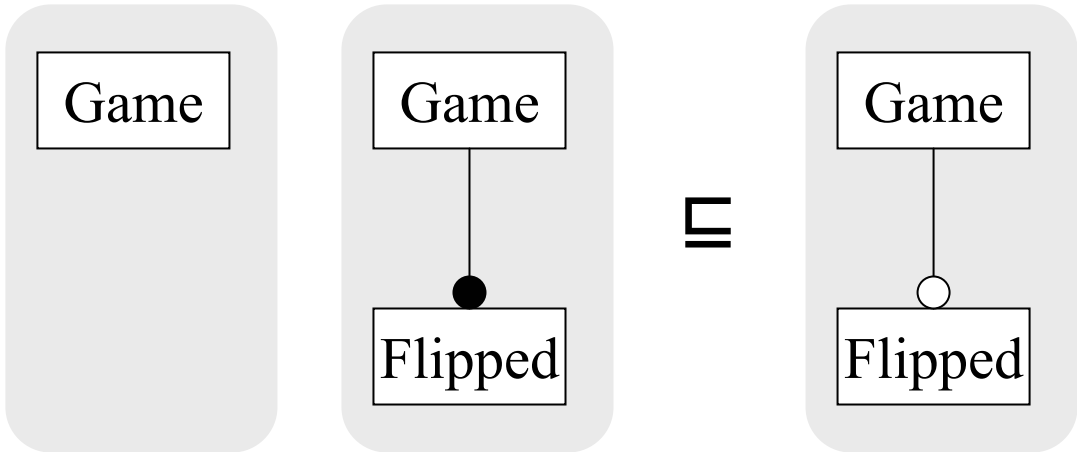


Proofs:

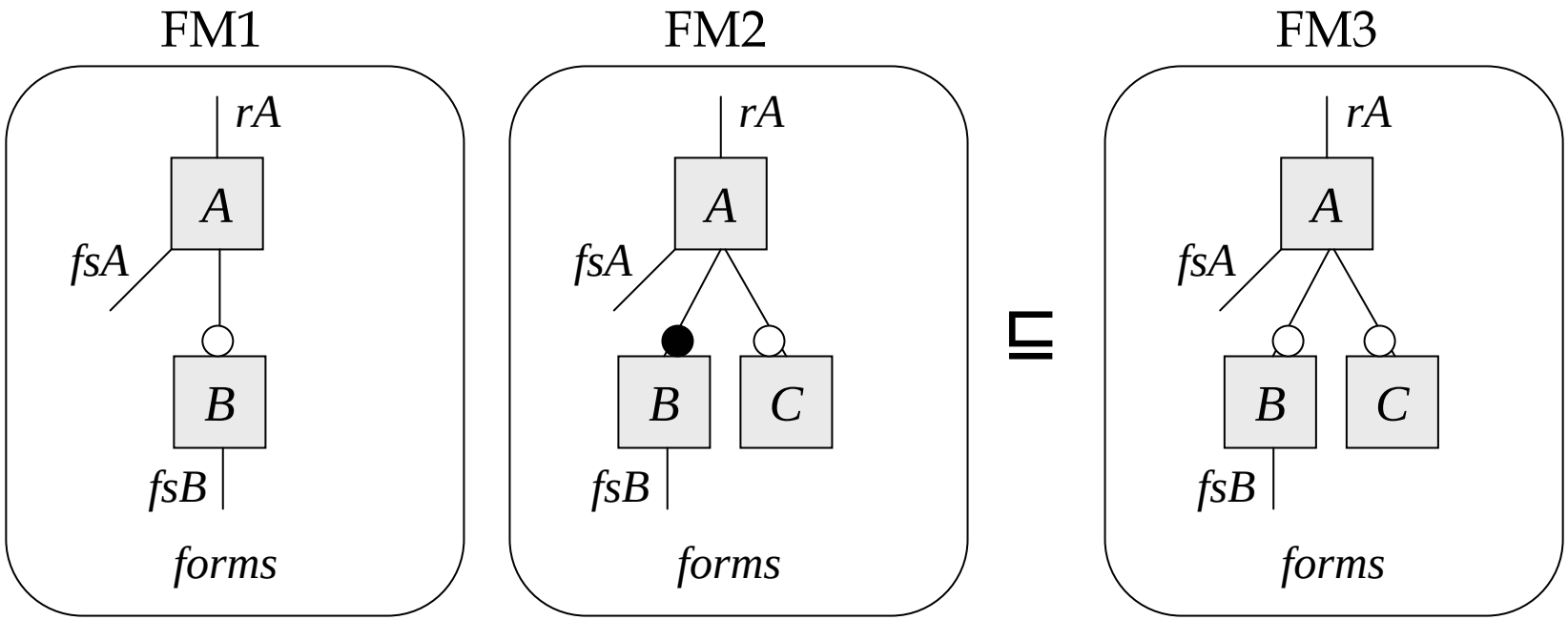
FM1 ⊑ FM3: Ref 12 (intro B)

FM2 ⊑ FM3: Ref 7 (B)

Example



Extractive 3

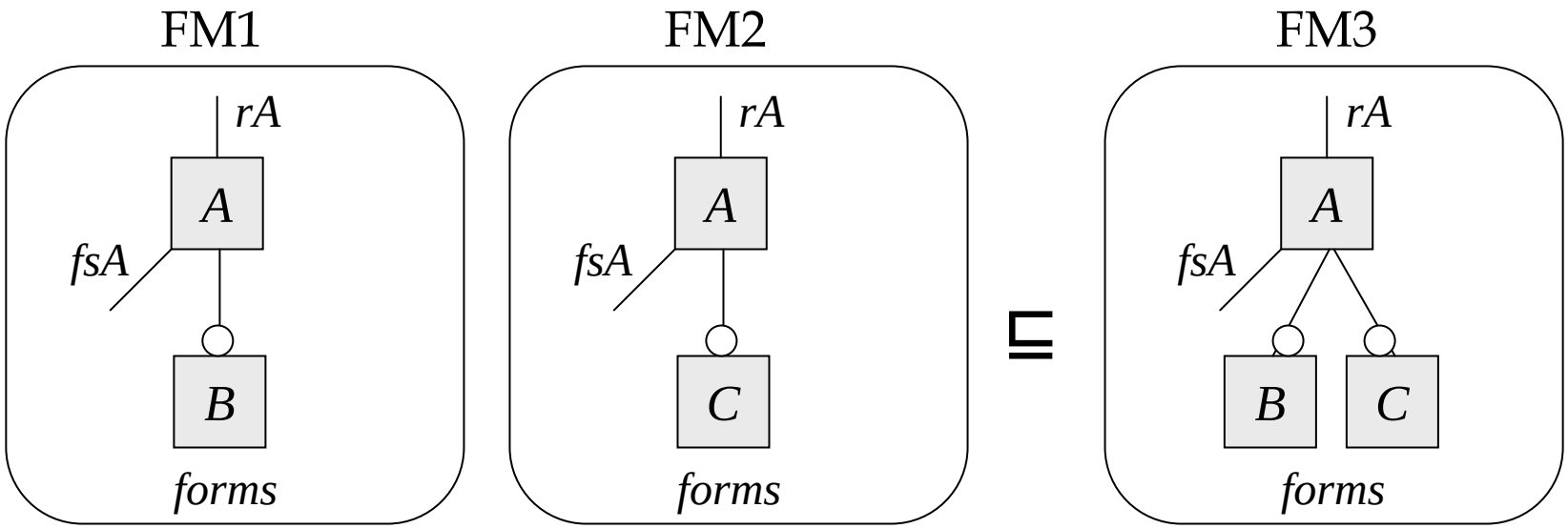


Proofs:

FM1 ⊑ FM3: Ref 12 (intro C)

FM2 ⊑ FM3: Ref 7 (B)

Extractive 4



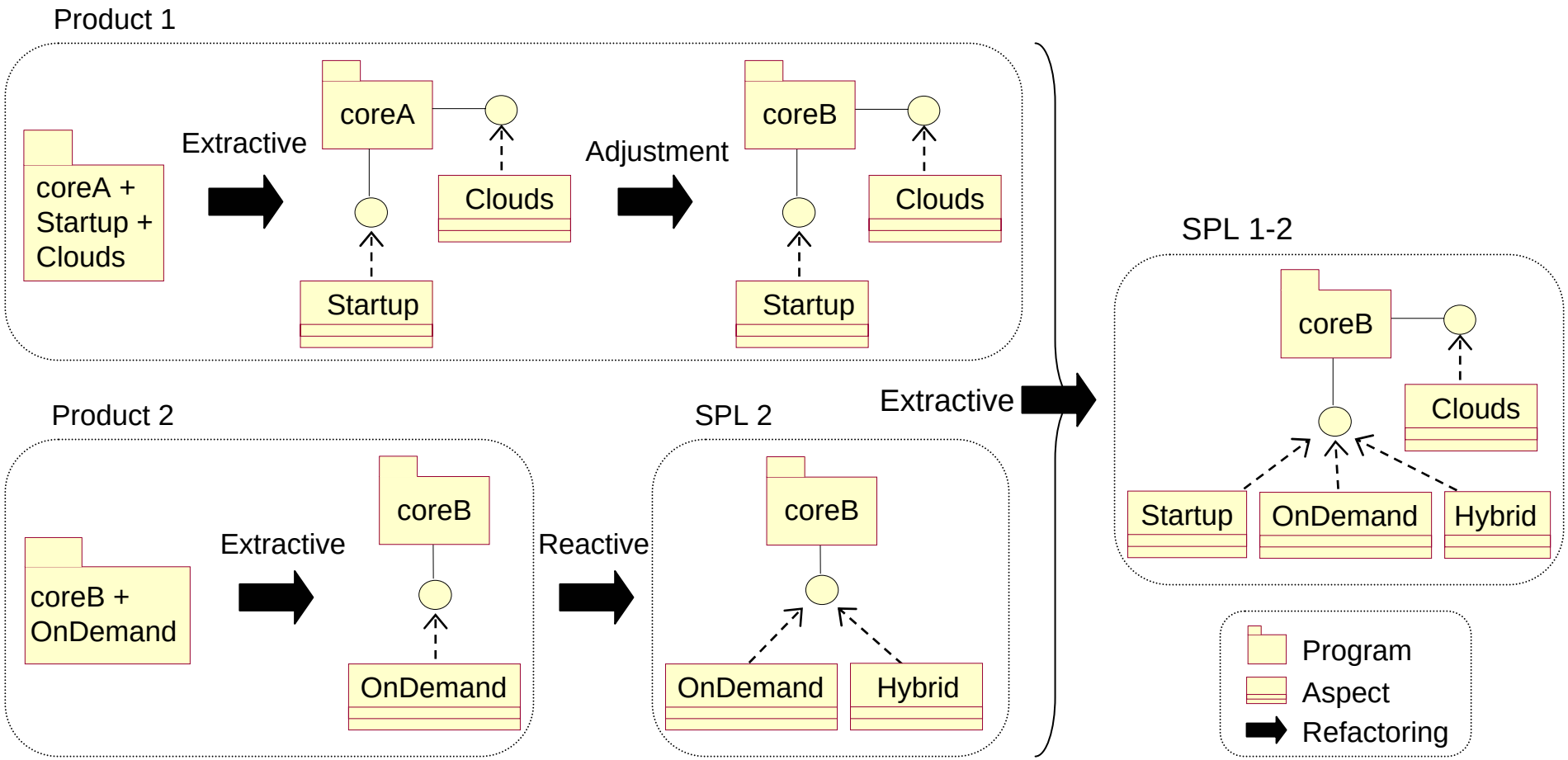
Proofs:

FM1 ⊑ FM3: Ref 12 (intro C)

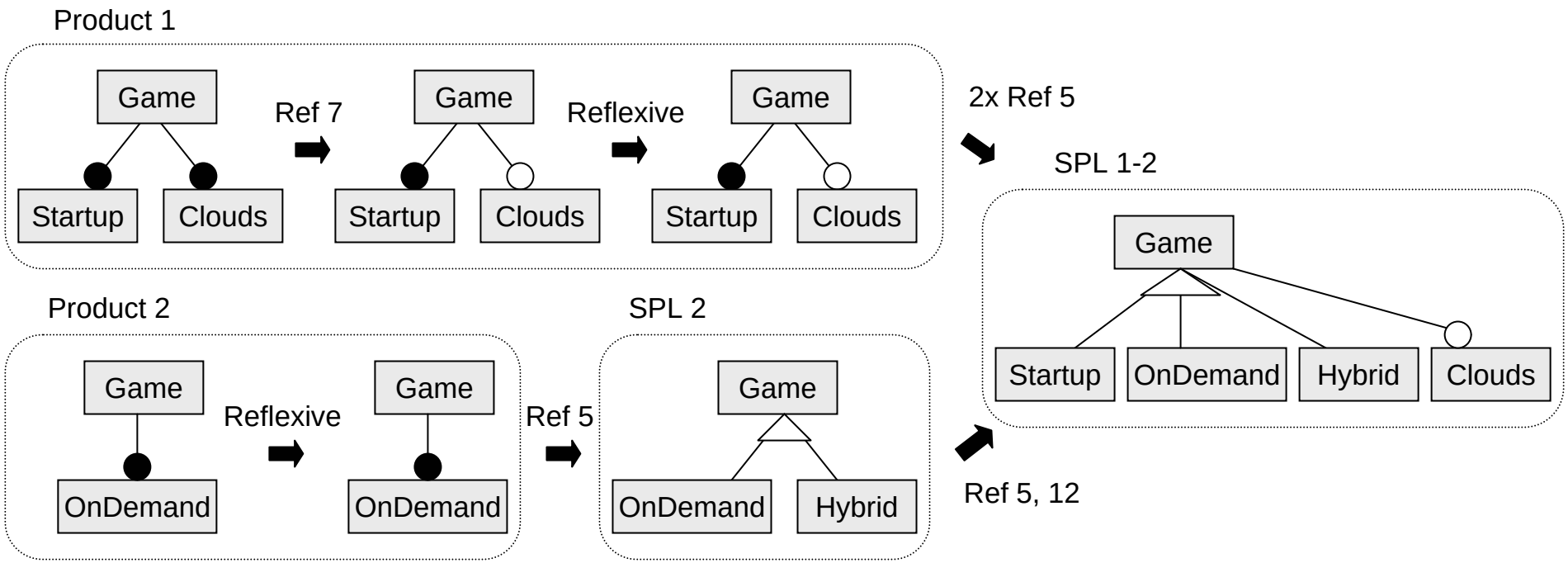
FM2 ⊑ FM3: Ref 12 (intro B)

Evaluation

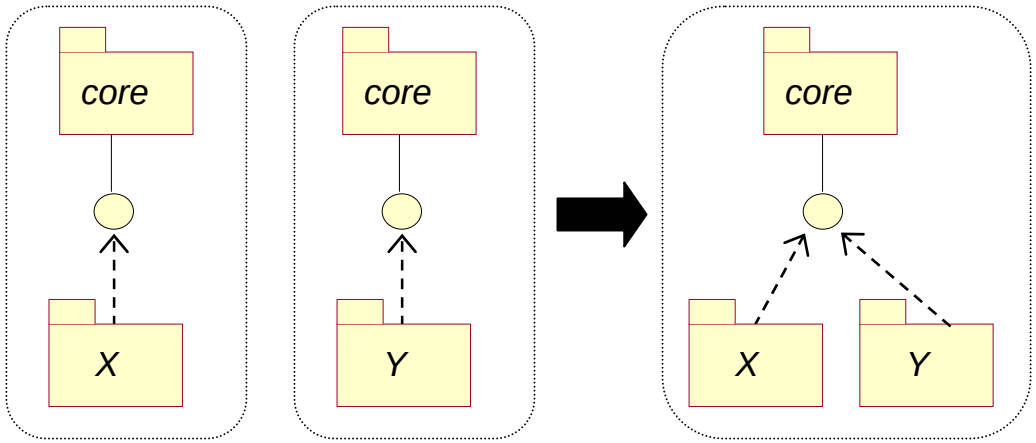
Case Study


Product 1
coreA + Startup + Clouds
Extractive
coreA
Clouds
Startup
Adjustment
coreB
Clouds
Startup
Product 2
coreB + OnDemand
Extractive
coreB
OnDemand
Reactive
SPL 2
coreB
OnDemand
Hybrid
Extractive
SPL 1-2
coreB
Clouds
Startup
OnDemand
Hybrid
Program
Aspect
Refactoring

FM Reasoning - Case Study



Product Line Refactoring 1



X and Y do not have any classes, aspects, members and methods with the same name