Once and for all: how to compose modules - the composition calculus

SummerSoc Thursday, June 27 2024

Peter Fettke

German Research Center for Artificial Intelligence (DFKI) and Saarland University, Saarbrücken, Germany

Wolfgang Reisig

Humboldt-Universität zu Berlin Germany

Prelude

Frank is smart.

Frank loves heavy math.

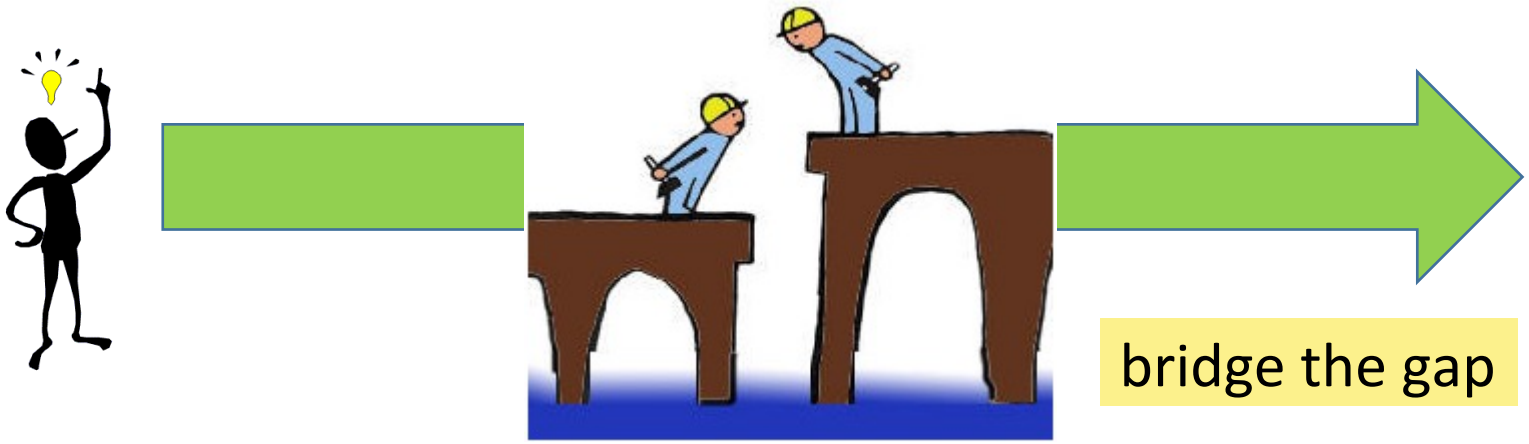
I love tiny, elegant math:

small amount of assumptions,

interesting consequences.

Software Engineering: a historical perspective

1968 NATO Conference on Software Engineering,
Garmisch Partenkirchen



bridge the gap

Software Engineering: a historical perspective

1968 NATO Conference on Software Engineering,
Garmisch Partenkirchen

bridge the gap

better!

Result: We need better programming languages!
Not just FORTRAN, COBOL, ALGOL 60

Monster Languages ALGOL 68, PL/1, ADA

tiny ones Pascal, Simula

Programming Paradigms

logic programs

functional programs

object orientation

Engineering!

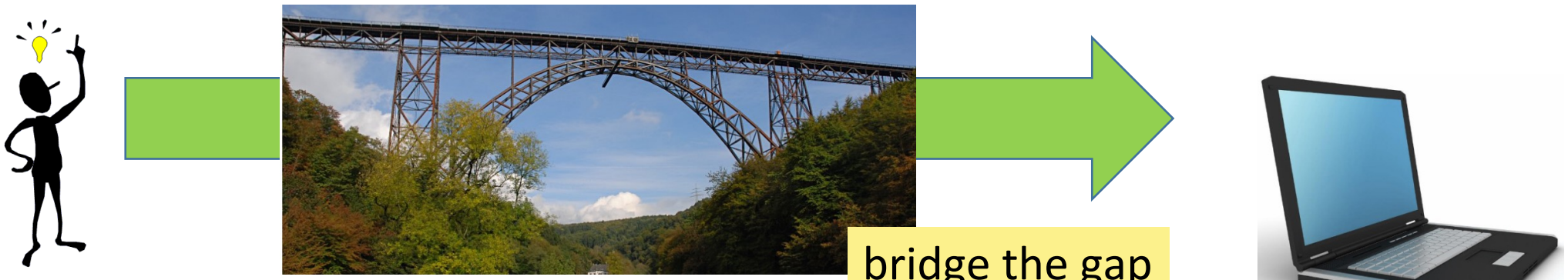
Software Engineering

Requirements Engineering

Software life cycles

Software Engineering: a historical perspective

1968 NATO Conference on Software Engineering,
Garmisch Partenkirchen



bridge the gap

better!

Software design methods

waterfall model

V-model

spiral model

agile methods

model based …

typical modeling techniques:

ARIS BPMN CASL EPK FOCUS
MSC STATECHARTS TLA UML Z

construct a model before you start coding!

a broader view: systems engineering

… either from the perspective of computing or remains informal

bridge the gap

better!

don‘t model just the software, but the entire system!

How to model systems?

So, what is a **good** modeling technique?

typical modeling techniques:

ARIS BPMN CASL EPK FOCUS MSC STATECHARTS TLA UML Z

… don’t really help!

ystem

a general deficit of classical modeling

a *good modell*

- separates components
- talks about real world items and data
- formulates behavior
- refines and abstracts modules as a dag, not a tree

… on any level of abstraction
… at any formal degree
as chosen by the modeler

a system

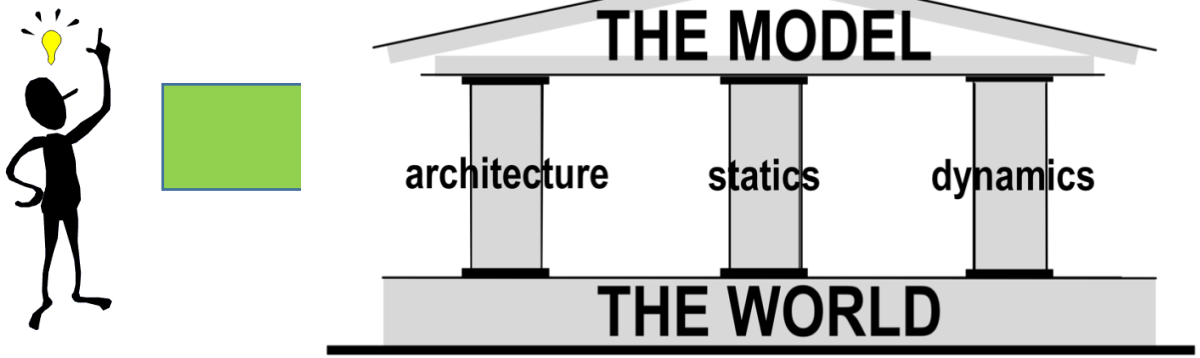
What a model is supposed to provide

with a model you can

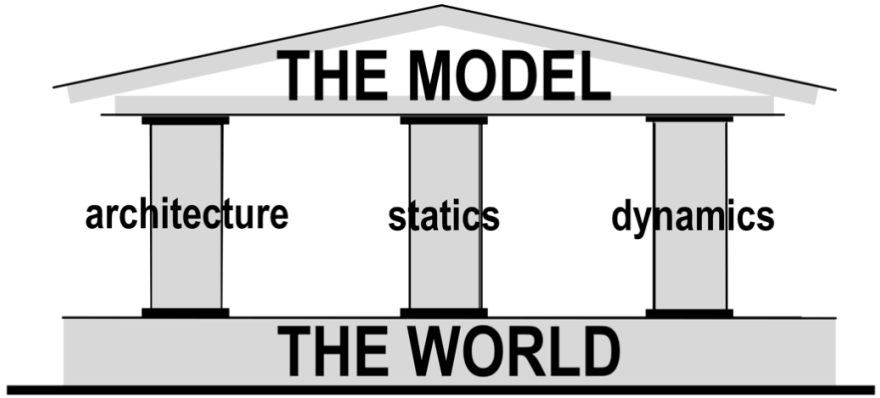
- formulate and prove system properties
- assess complexity of behavior
- explain a system to stake holders
- estimate costs
- formulate agreements and orders
- specify functionalities and warranties

a system

First assumptions ..



a system



today: only about architecture

Peter Fettke · Wolfgang Reisig

Understanding the Digital World

Modeling with HERAKLIT

This book fills a serious gap by providing a conceptual framework for understanding the digital world. This world contains large, heterogeneous systems that have to manage dynamic behavior as well as static items and data. Obviously, new, *digital methods* are needed to deal with the challenges of the digital world.

This book introduces such a method with HERAKLIT, an intuitively simple, albeit powerful framework for modeling, communicating, and analyzing computer-

nodules, describing

eal- and imagined-

technically simple,

, starting in Part I

their composition

ynamics, focusing

n static aspects. In

representation are

s are consolidated

nework. The book

ical retail business,

d useful graphical

els.

eas for a computer-

s, the contributions

uction of software.

system modeling,

als in these fields.

ISBN 978-3-031-61897-0

▶ springer.com

Fettke · Reisig

Understanding the Digital World

Peter Fettke
Wolfgang Reisig

Understanding the Digital World

Modeling with HERAKLIT

Springer

Once and for all: how to compose modules - the composition calculus

What justifies this talk's title ?

(i) Case studies in various domains

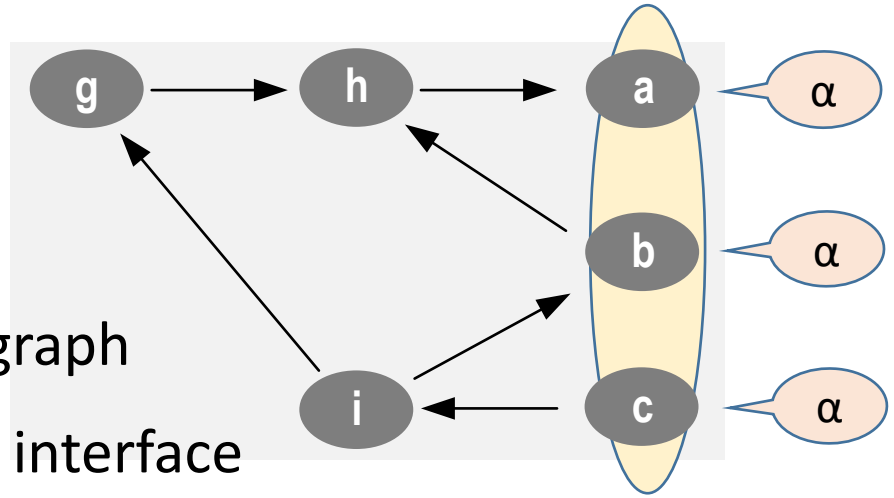
(ii) Innovative applications, e.g. large process models *Next talk, Peter*

(iii) Generality of the composition calculus

- A module is a graph. There is nothing more general than graphs.
- An interface is a set of vertices – labeled, ordered. Composition needs nothing more.
- Composition is technically simple – and total.

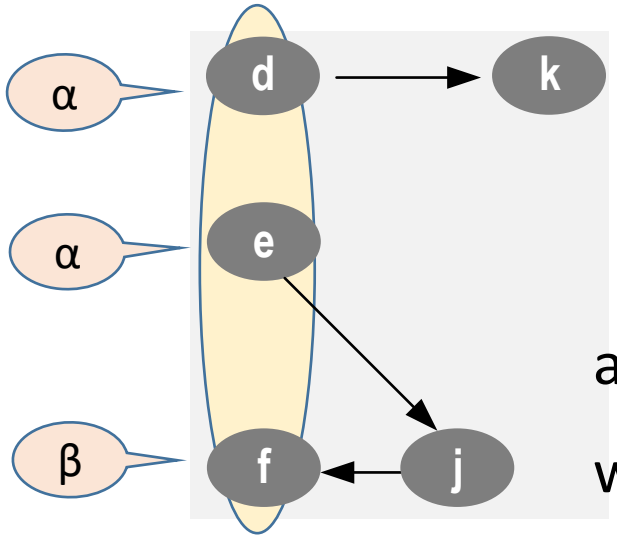
(iv) Mathematically deep and practically useful properties: *This talk*

1. Modules and Their Composition



a graph

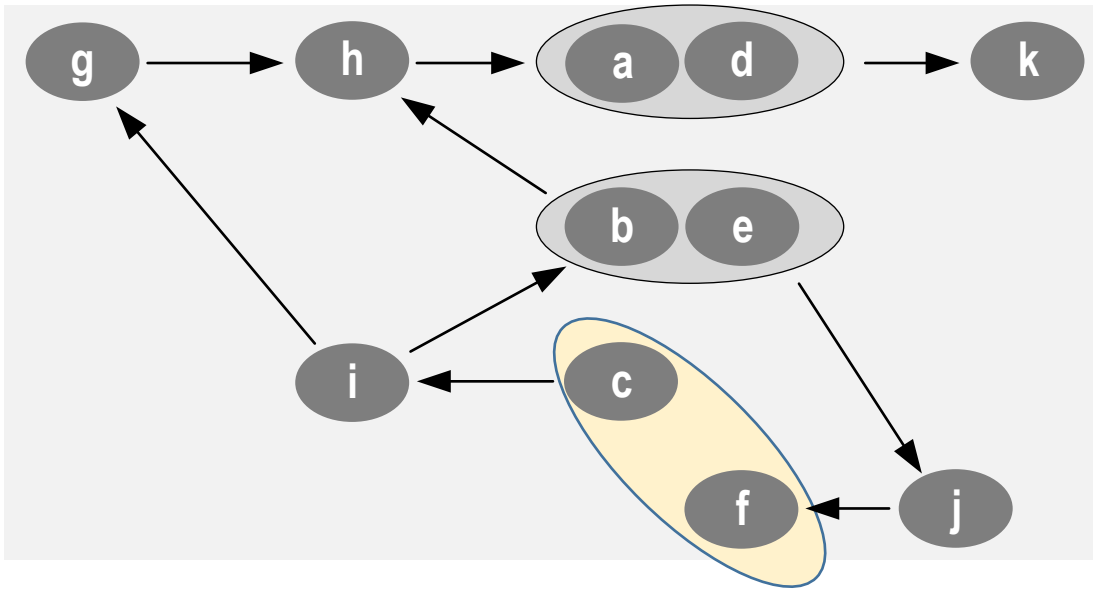
an interface



another graph

with interface

composition:



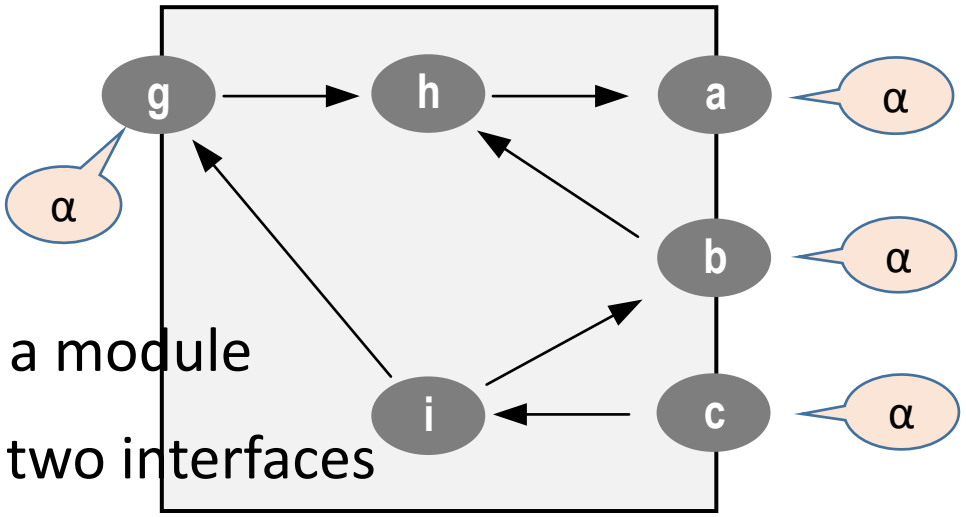
Merge equally labeled gates

along the order of the interfaces.

What to do with c and f?

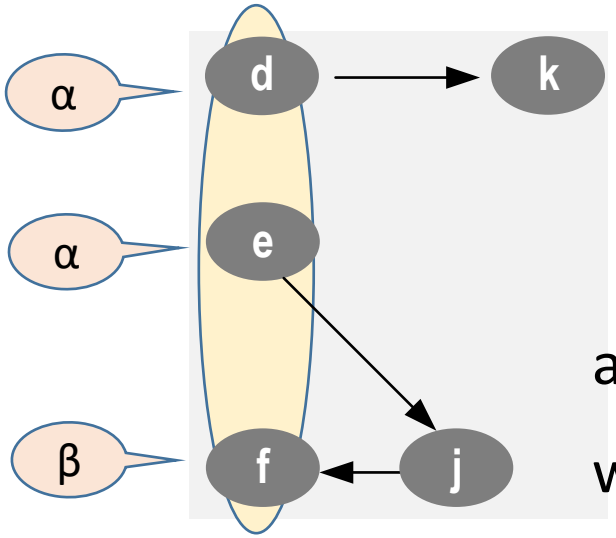
Go to the interface

... another try



a module

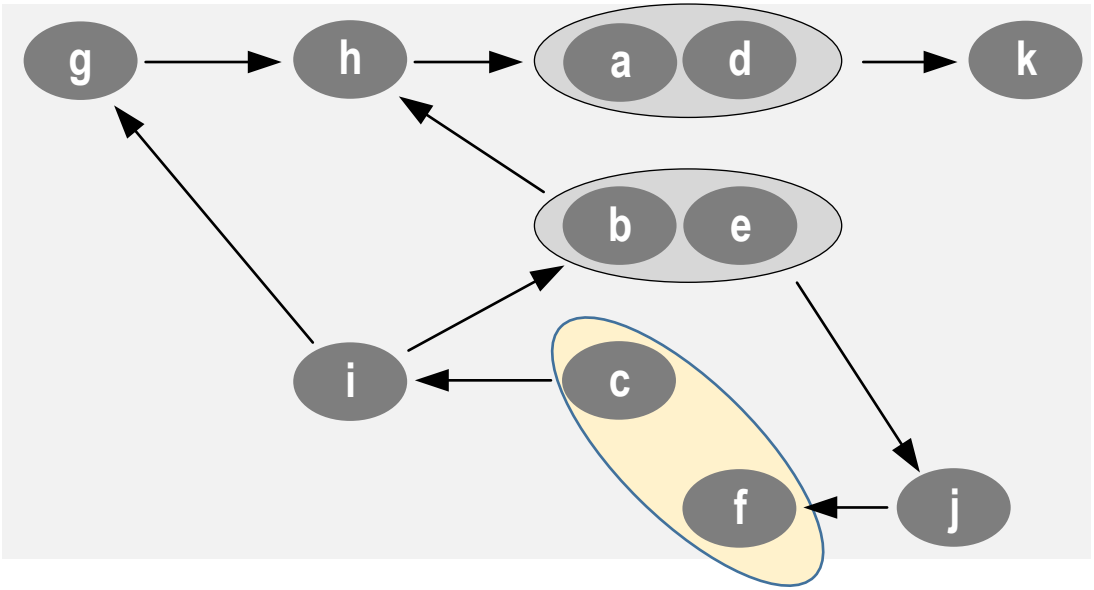
two interfaces



another graph

with interface

composition:



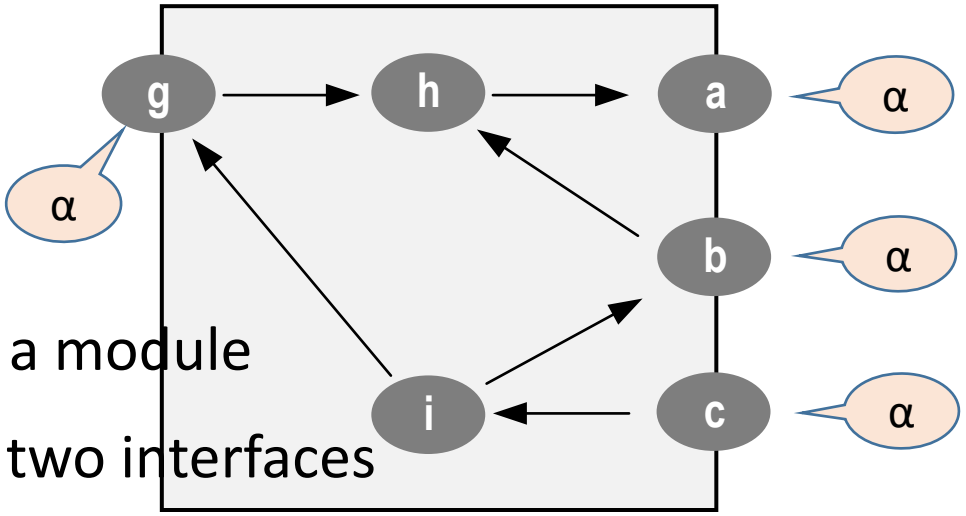
Merge equally labeled gates

along the order of the interfaces.

What to do with c and f?

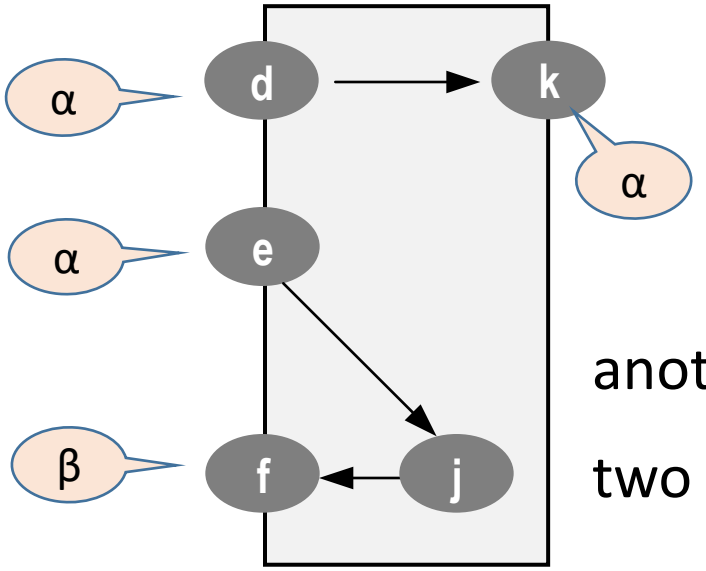
Go to the interface

... another try



a module

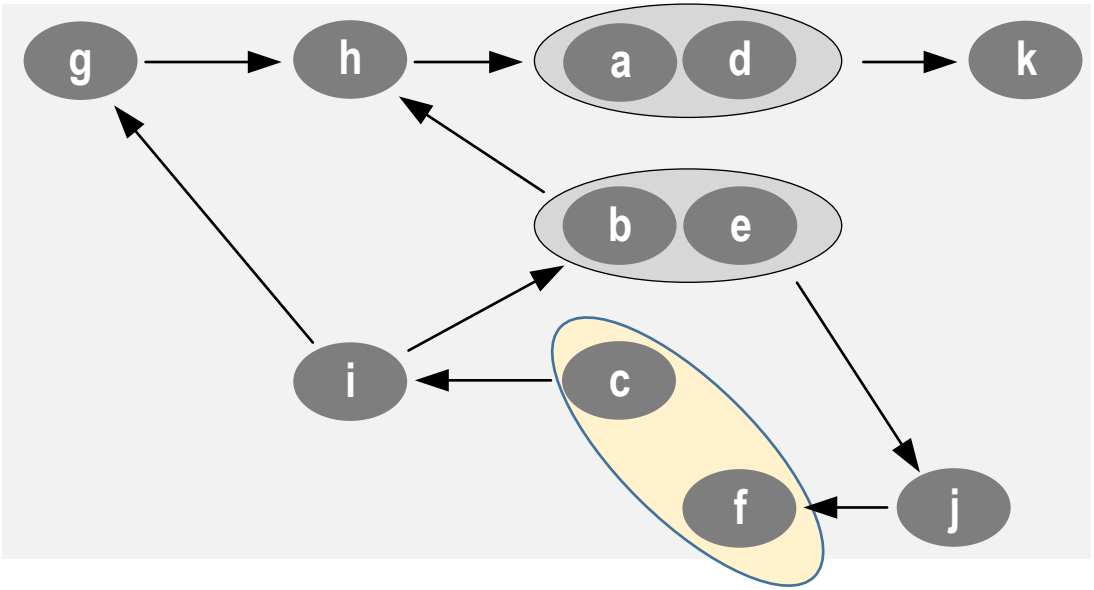
two interfaces



another module

two interfaces

composition:



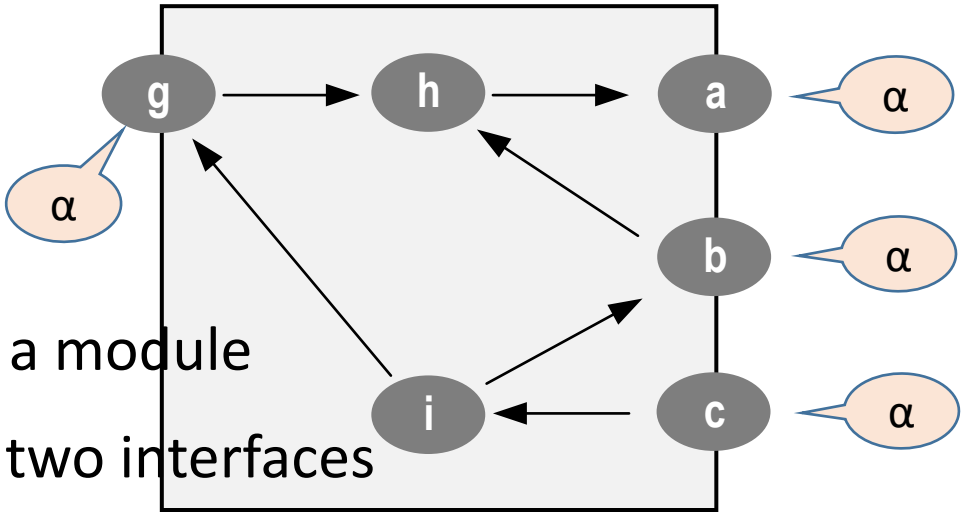
Merge equally labeled gates

along the order of the interfaces.

What to do with c and f?

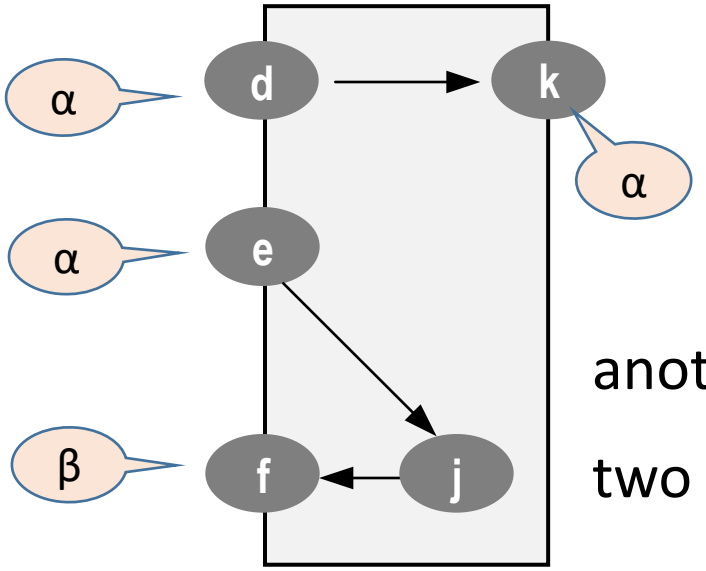
Go to the interface

... another try



a module

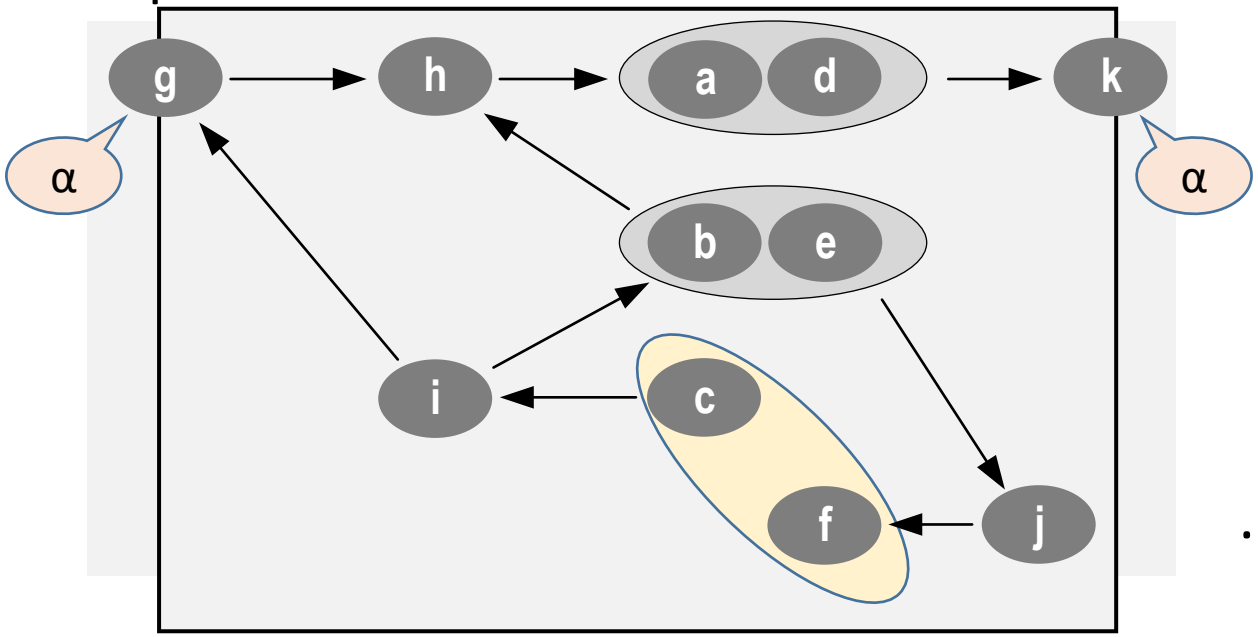
two interfaces



another module

two interfaces

composition:



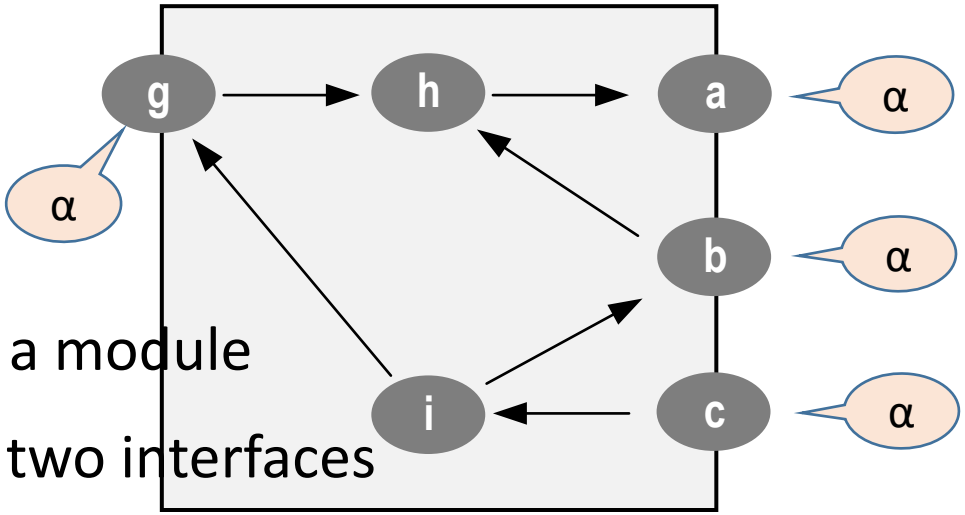
Merge equally labeled gates

along the order of the interfaces.

What to do with c and f?

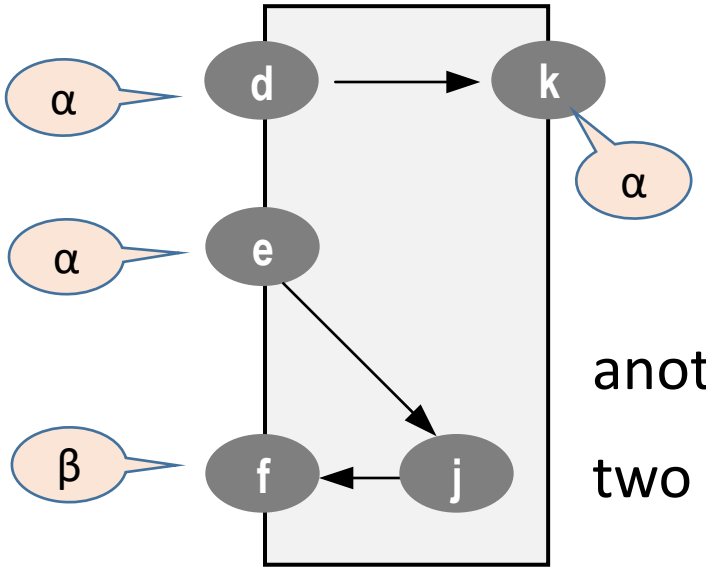
Go to the interface

... another try



a module

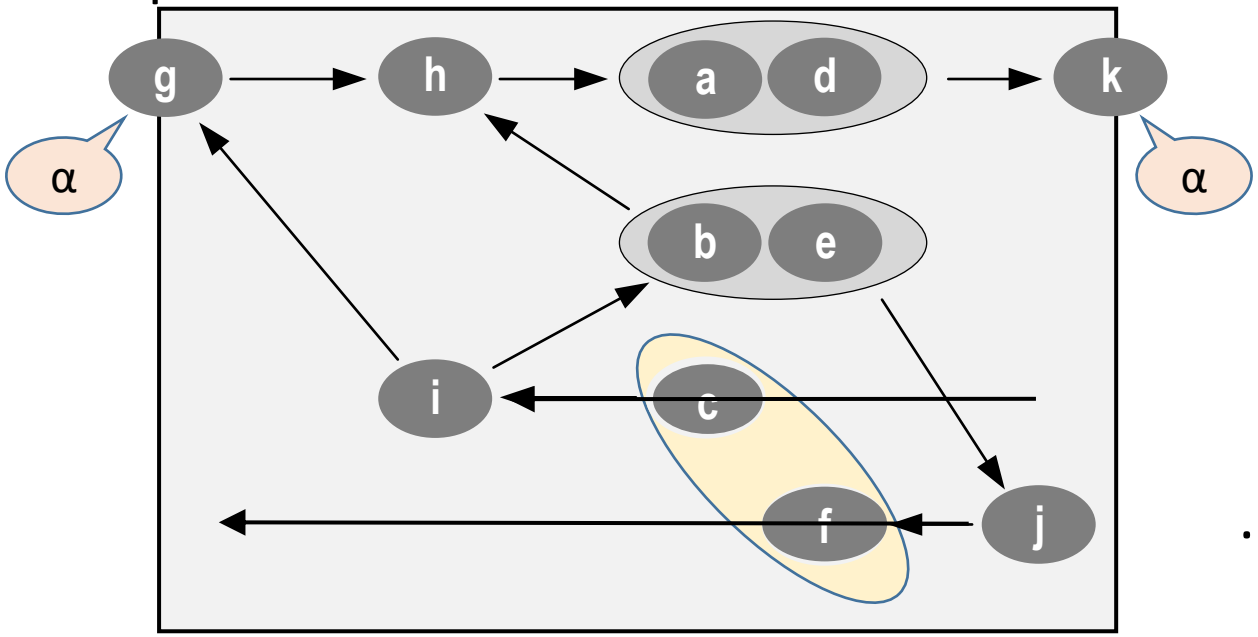
two interfaces



another module

two interfaces

composition:



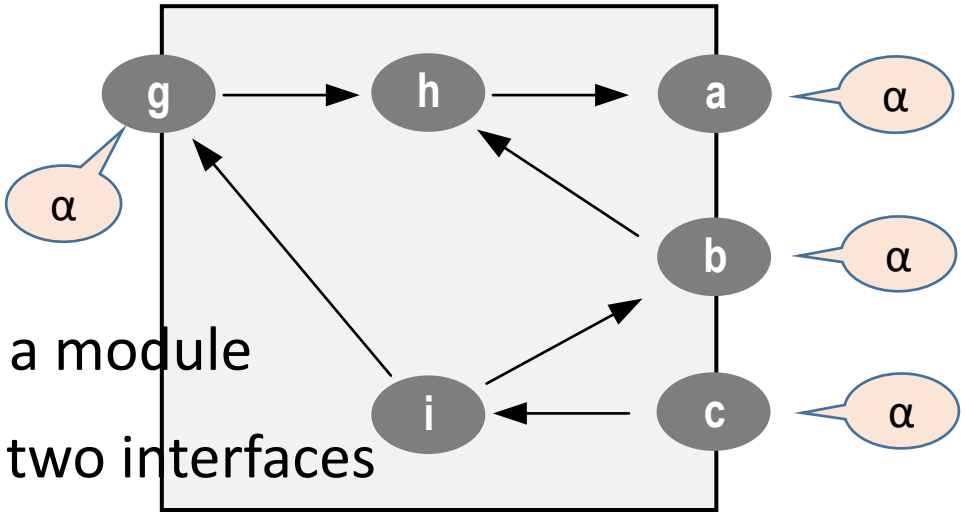
Merge equally labeled gates

along the order of the interfaces.

What to do with c and f?

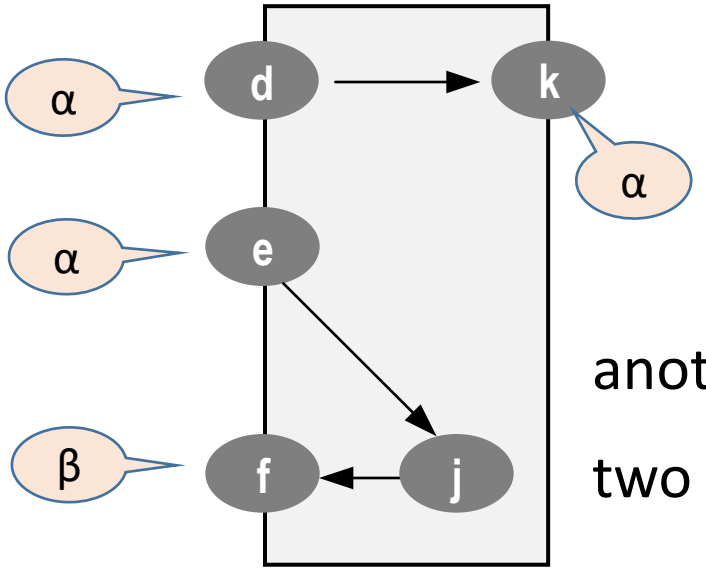
Go to the interface

... another try



a module

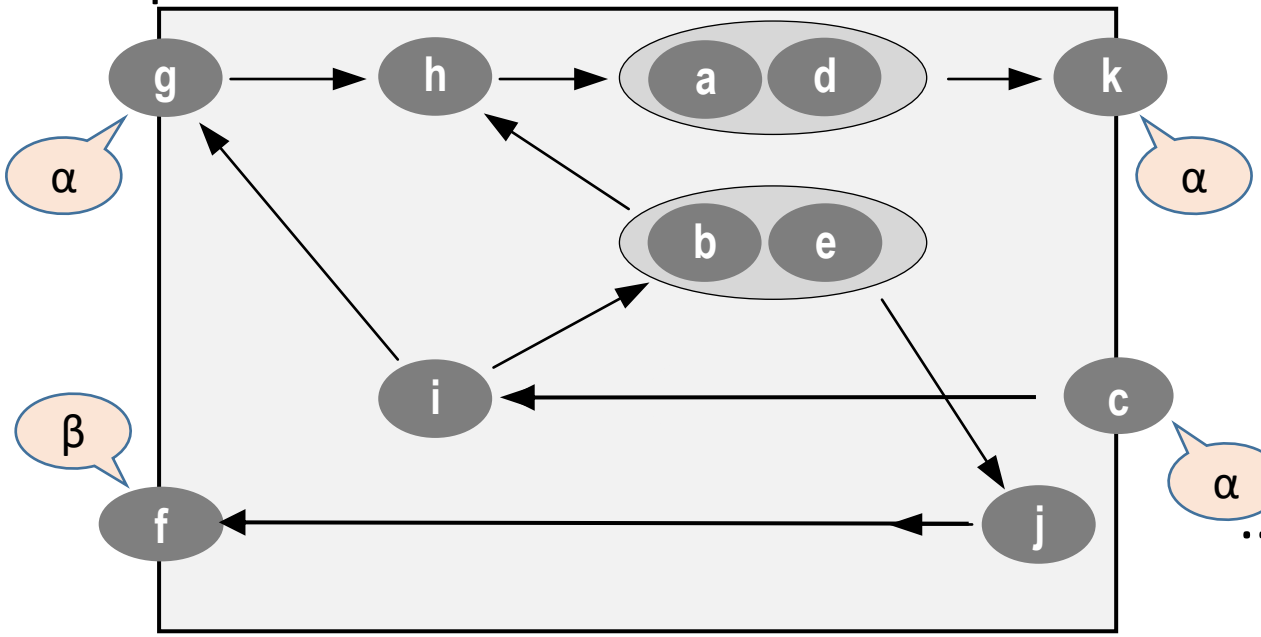
two interfaces



another module

two interfaces

composition:



Merge equally labeled gates

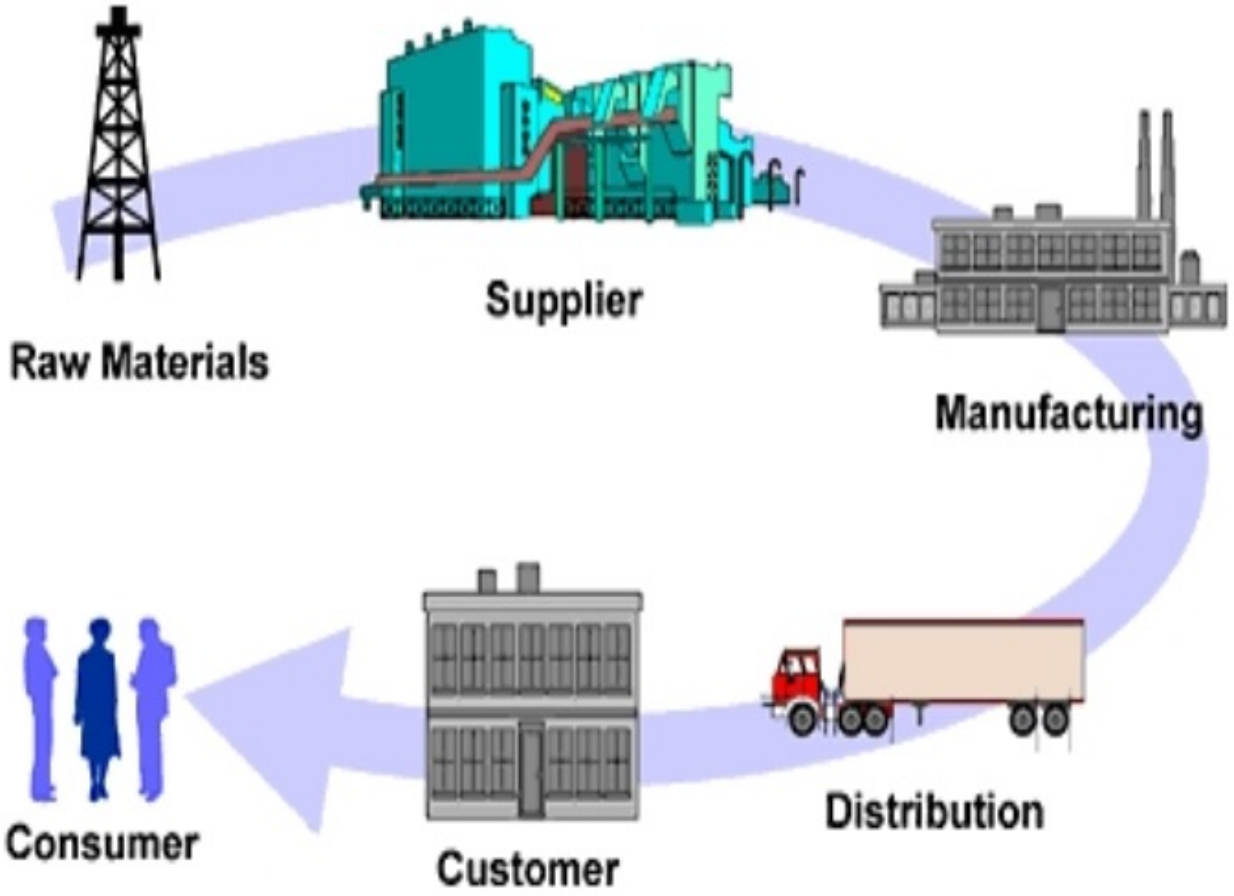
along the order of the interfaces.

What to do with c and f?

Go to the interface

... another try

two interfaces: most natural



written:

RM • Su • Ma • Di • Cu • Co

- ***input*** and ***output***,
- ***customer*** and ***supplier***,
- ***requester*** and ***provider***,
- ***consumer*** and ***producer***,
- ***buy side*** and ***sell side***,
- ***predecessor*** and ***successor***,
- ***assumptions*** and ***guarantees***,
- ***pull*** and ***push***,
- ***left*** and ***right***,

etc.

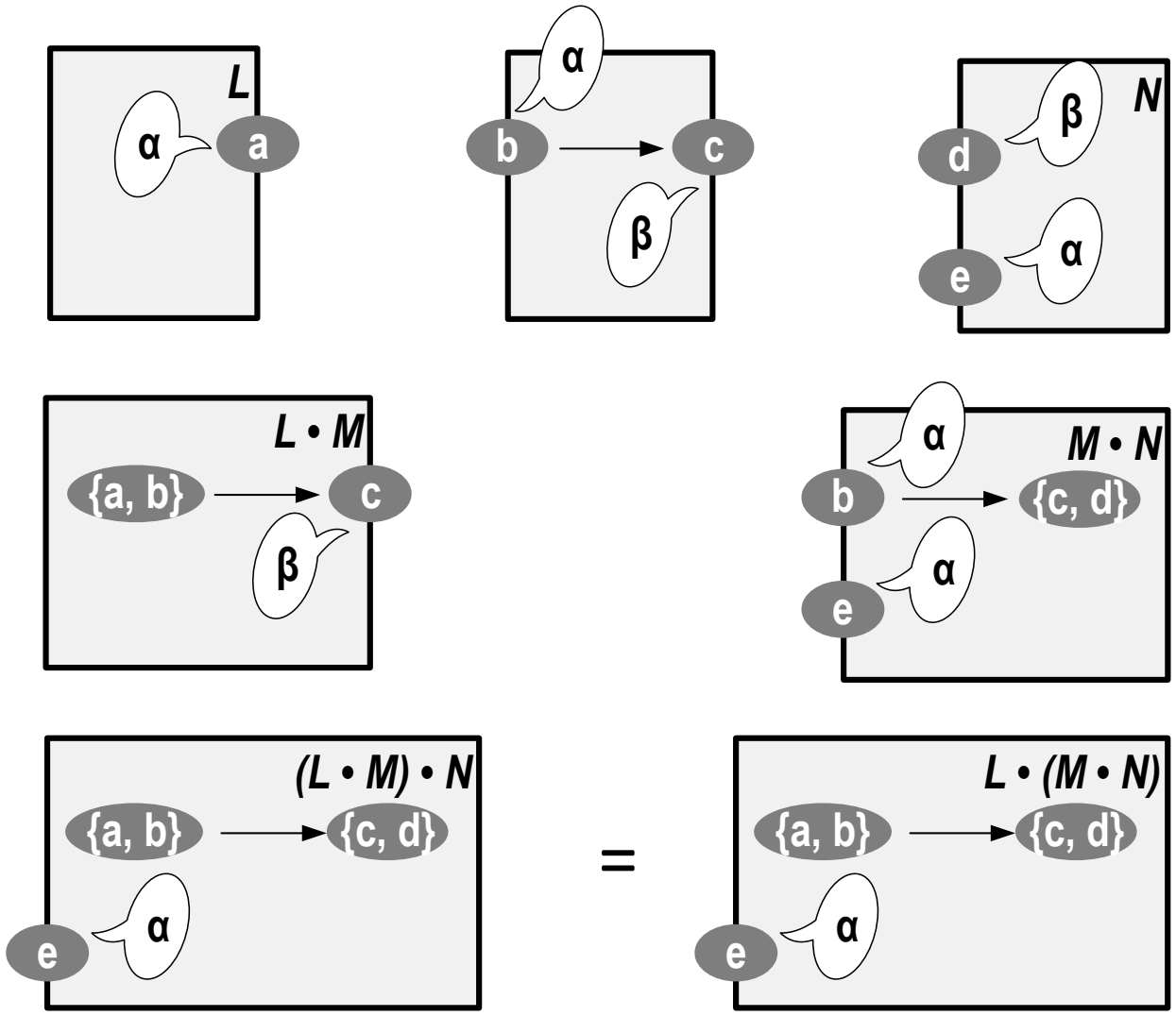
2. Algebraic properties of modules
2.1 Associativity

Theorem Let L, M, and N be three modules. Then

$$(L \bullet M) \bullet N = L \bullet (M \bullet N).$$

Just write $M_1 \bullet M_2 \bullet \ldots \bullet M_n$.

2. Algebraic properties of modules
2.1 Associativity



2. Algebraic properties of modules
2.2 Cancelativity

M • N: like scrambled eggs, or like composed words over an alphabet?

Theorem Let L, M, N be modules.

If L • M = L • N, then M = N.

If M • L = N • L, then M = N.

... hence, like words over an alphabet!

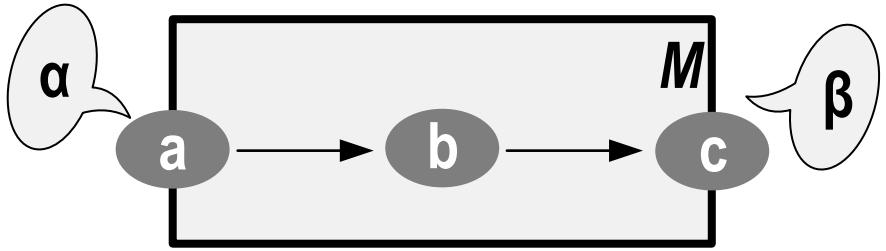
2. Algebraic properties of modules
2.3 Commutativity

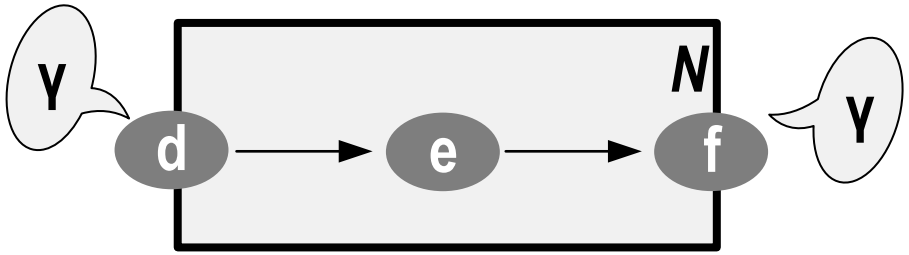
When holds: M • N = N • M?

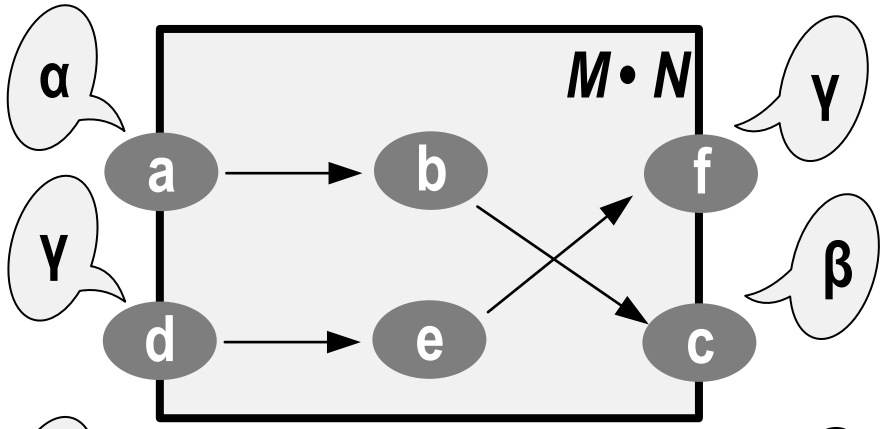
Theorem M • N and N • M are equivalent iff no label occurs in the interfaces of M as well as in the interfaces of N.

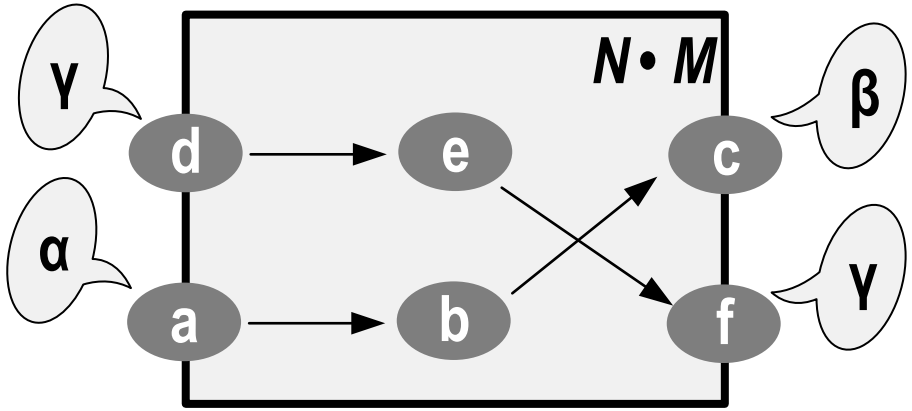
useful for paramererized modules $M_1, \ldots, M_n$

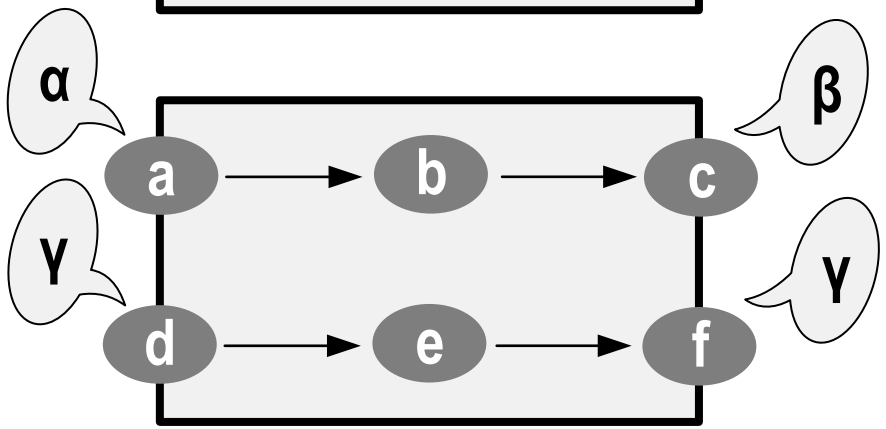
2. Algebraic properties of modules
2.3 Commutativity

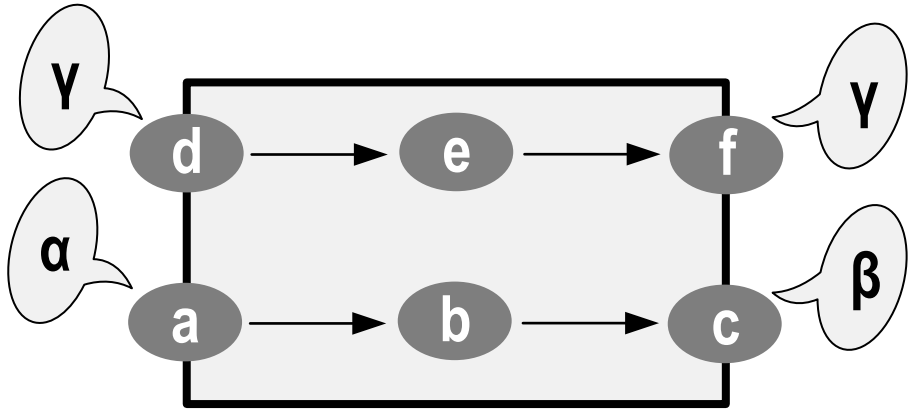




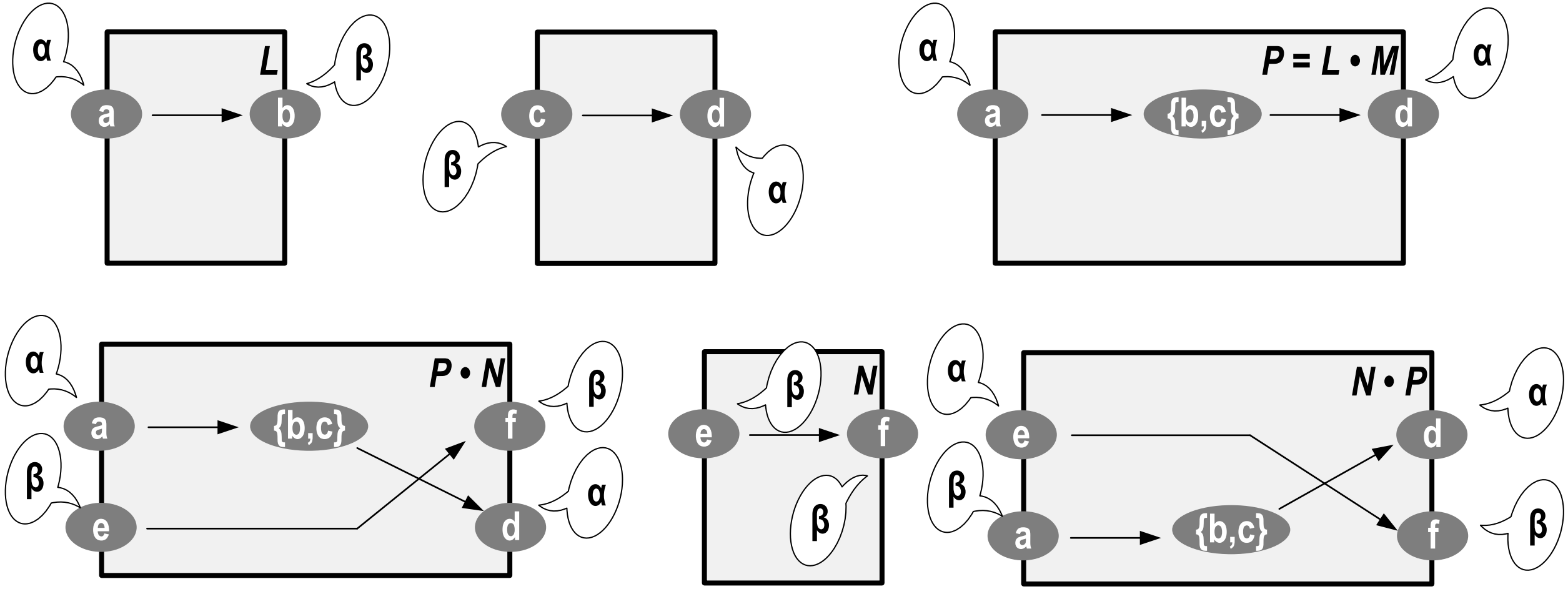








2. Algebraic properties of modules
2.3 Commutativity

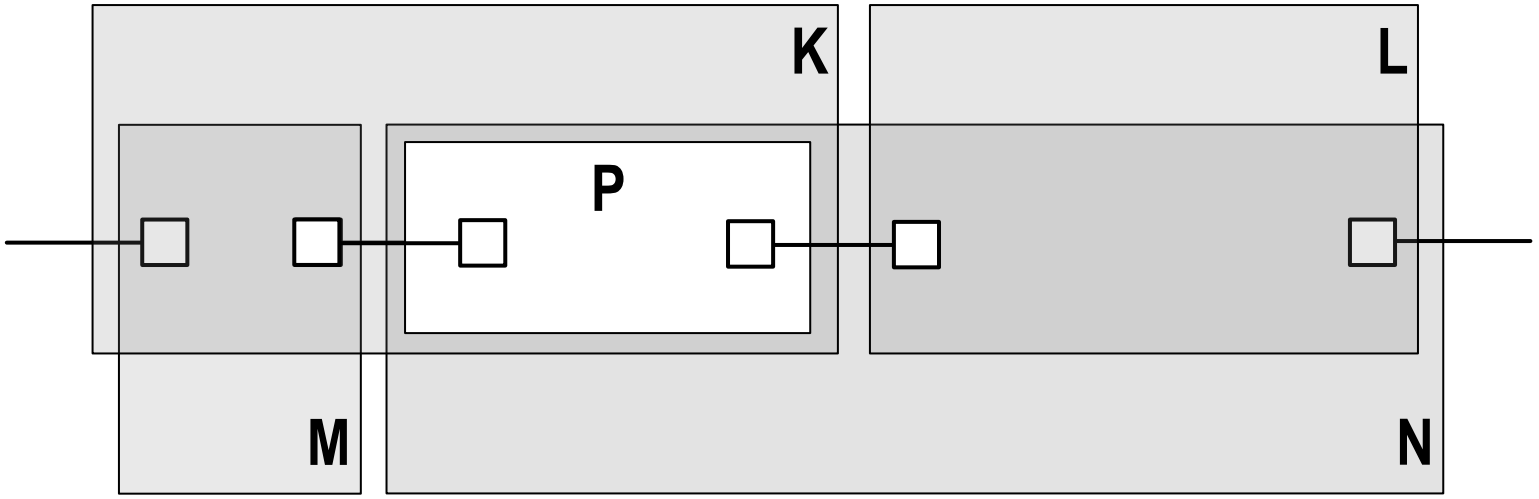


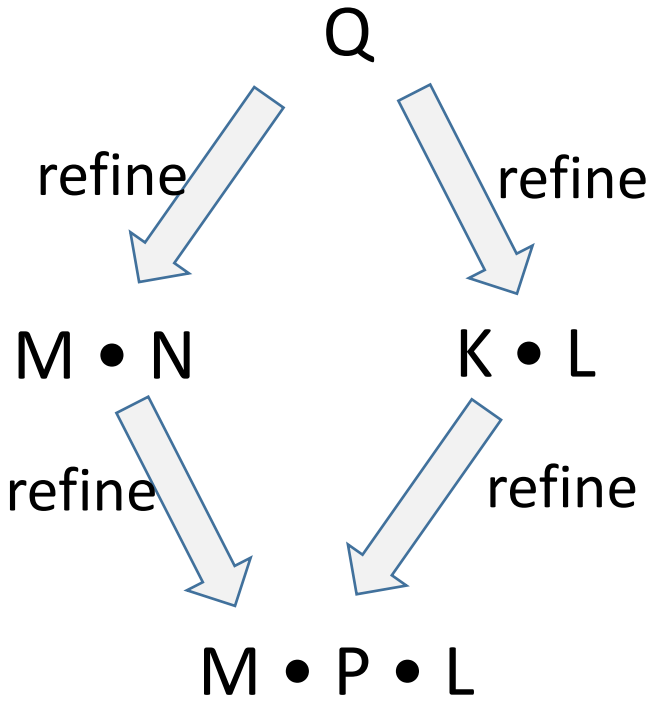
not commuting: L---N and M---N. But commuting: (L • M)---N

2. Algebraic properties of modules
2.4 Equidivisibility

Assume Q = M • N = K • L, not commuting

Q:





Then there exists a unique module P such that

Q = M • P • L

3. Distinguished interfaces
3.1 General Modules

The module $E =_{def} (\emptyset, \emptyset)$ is denoted as *zero module*

Lemma For each module M holds: $M \bullet E = E \bullet M = M$.

Theorem Let *Mod* be the set of all modules over Λ.
Then $mod(\Lambda) =_{def} (Mod, \bullet, E)$ is a monoid.

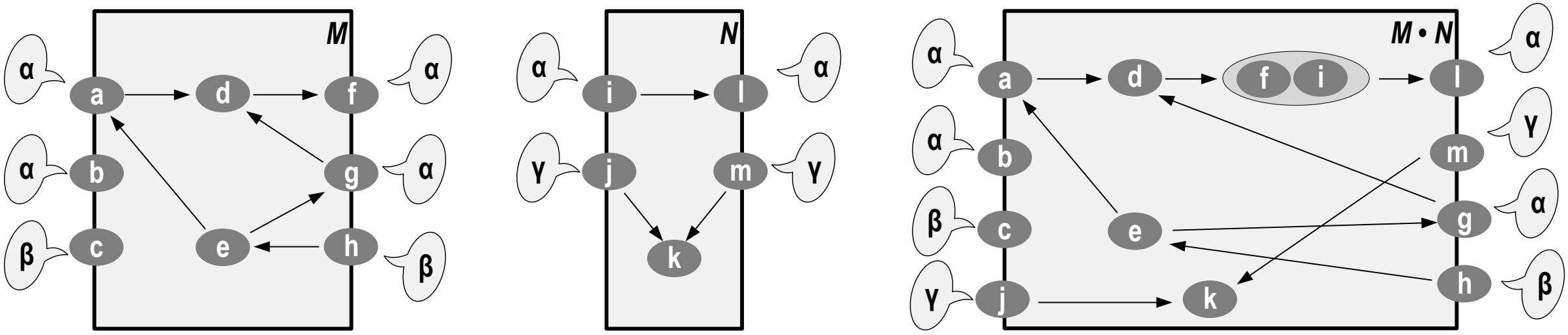
Theorem Let $\Delta \subseteq \Lambda$. Then $mod(\Delta)$ is a submonoid of $mod(\Lambda)$.

3. Distinguished interfaces
3.2 Cyclic modules

Def.: (i) A module M is *cyclic*, if *M and M* are equivalent.

Let *cyclic*(Λ) denote the set of all cyclic modules over Λ.

Theorem *cyclic*(Λ) is a submonoid of mod(Λ).

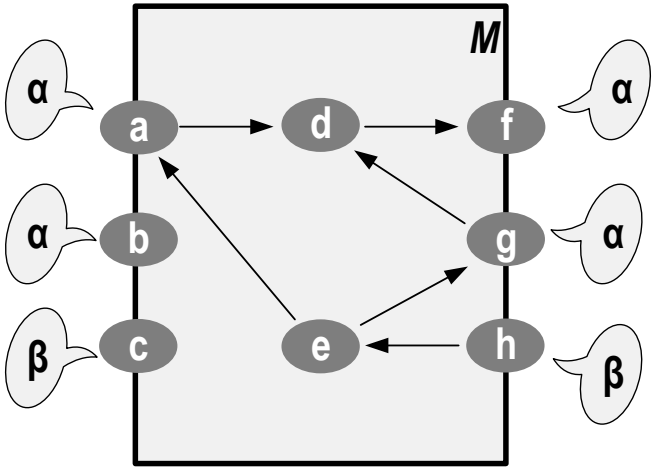


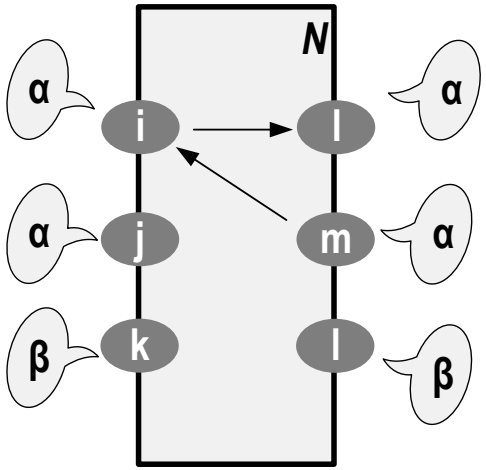
3. Distinguished interfaces
3.3 Fixed interface modules

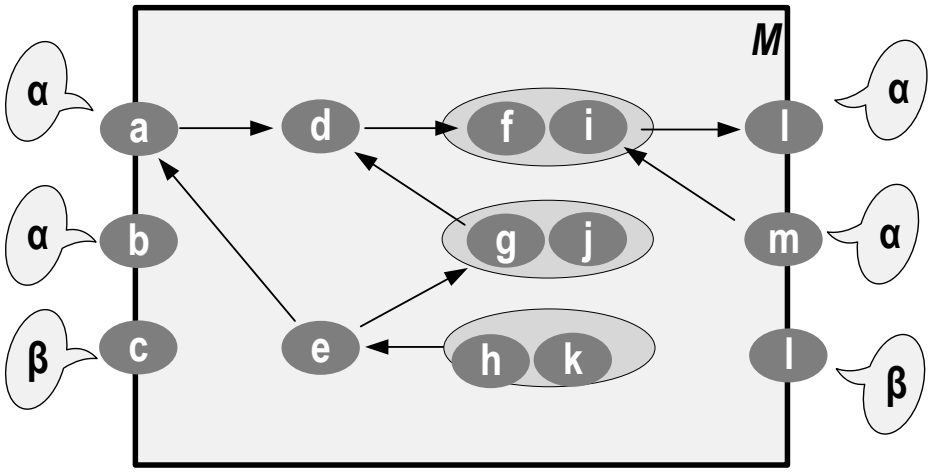
Def.: Let X be an interface over Λ. *fixed*(X): all modules M with

- M = E or
- *M as well as M* equivalent to X.

Theorem *fixed*(Λ) is a submonoid of *cyclic*(Λ).

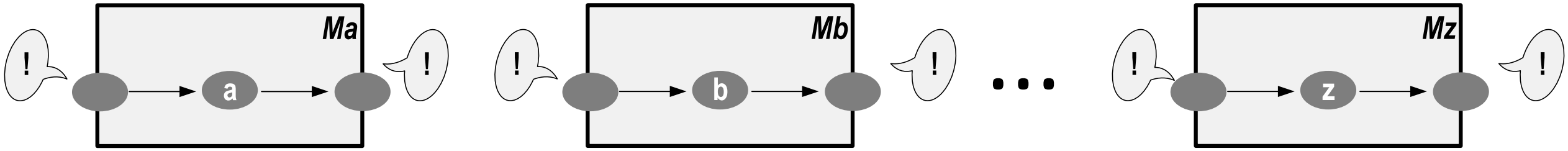






4. Distinguished interior
4.1 Alphabet modules

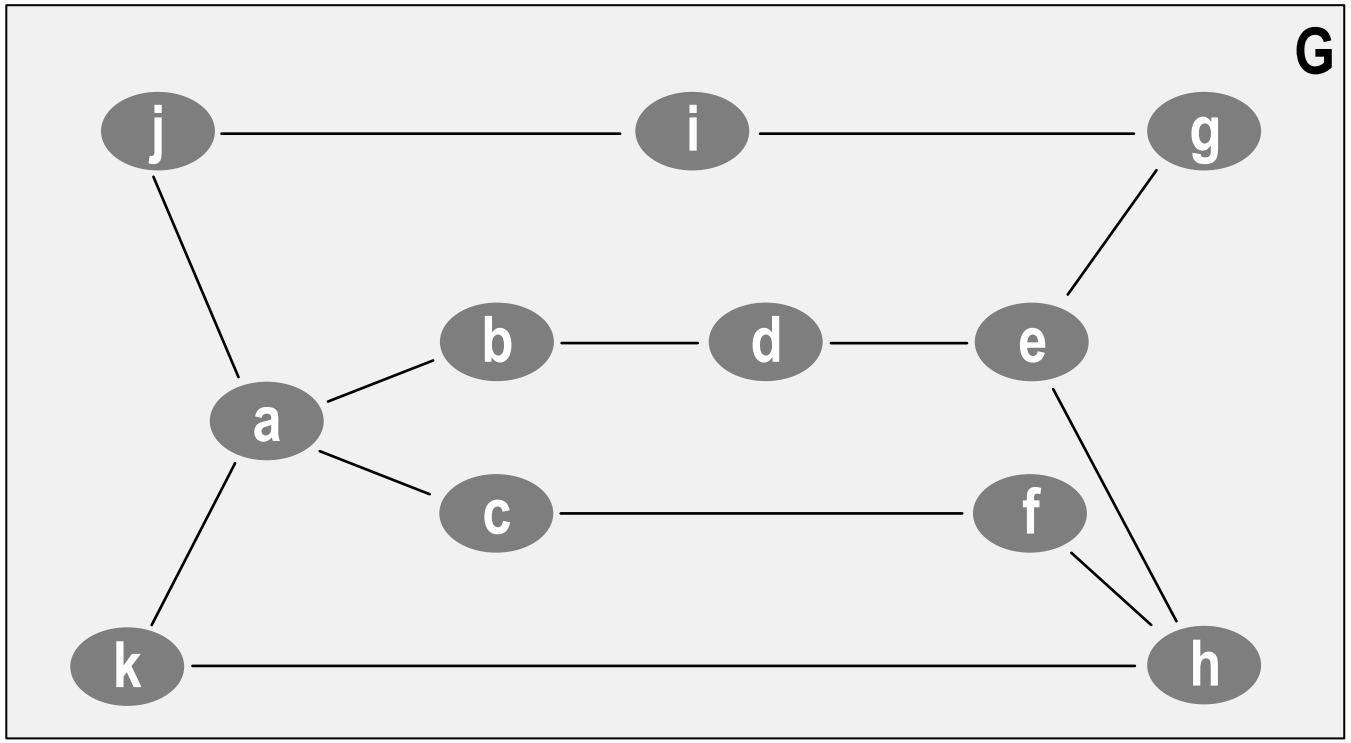
$\Sigma = \{a, \ldots, z\}$



the free monoid of words over Σ,
formal languages,
conventional informatics.

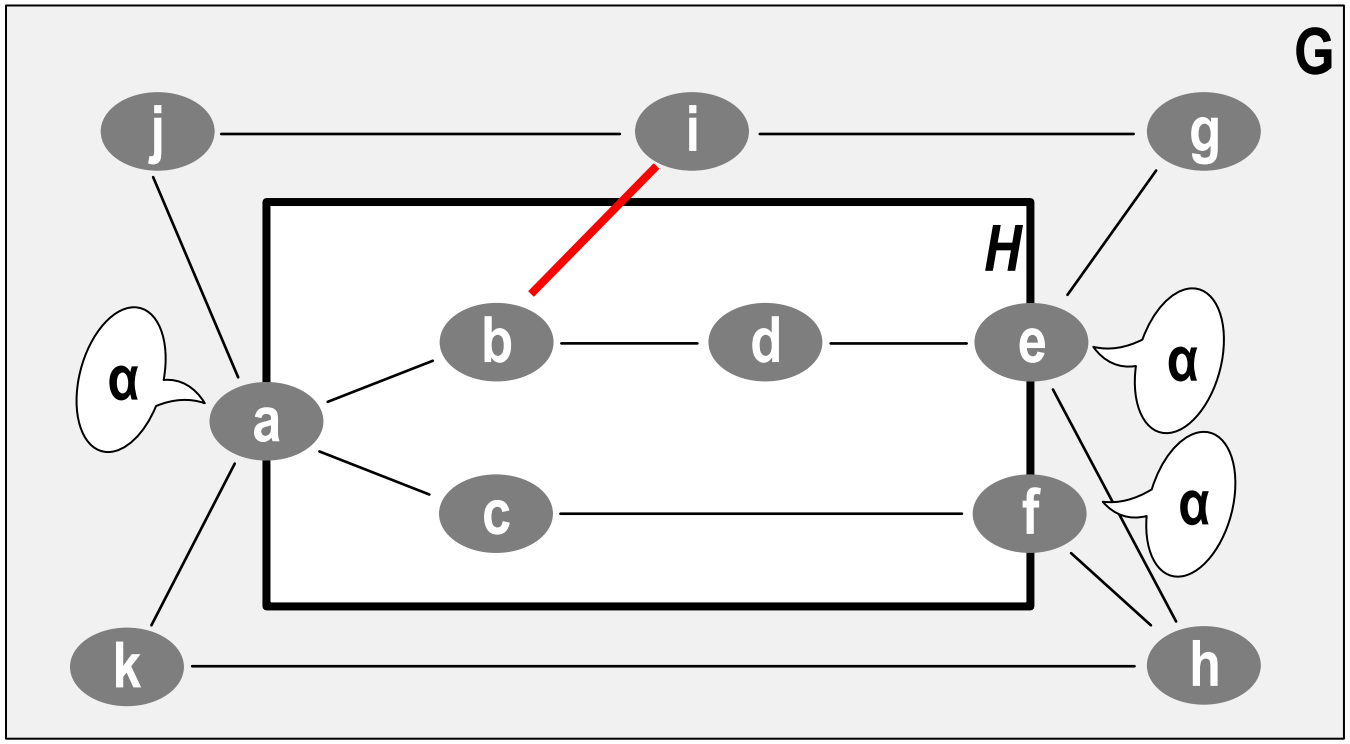
4. Distinguished interior
4.2 submodules

a graph …



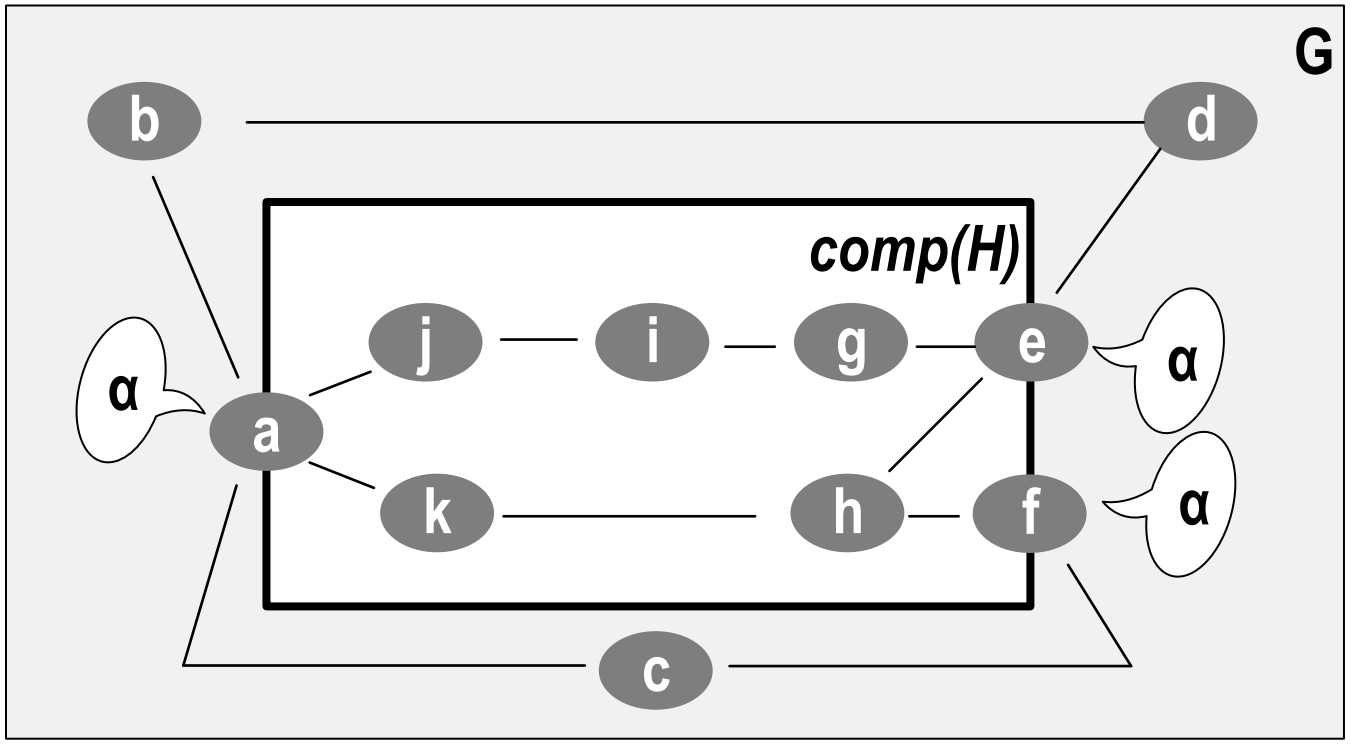
4. Distinguished interior
4.2 submodules

a submodule, H no more a submodule



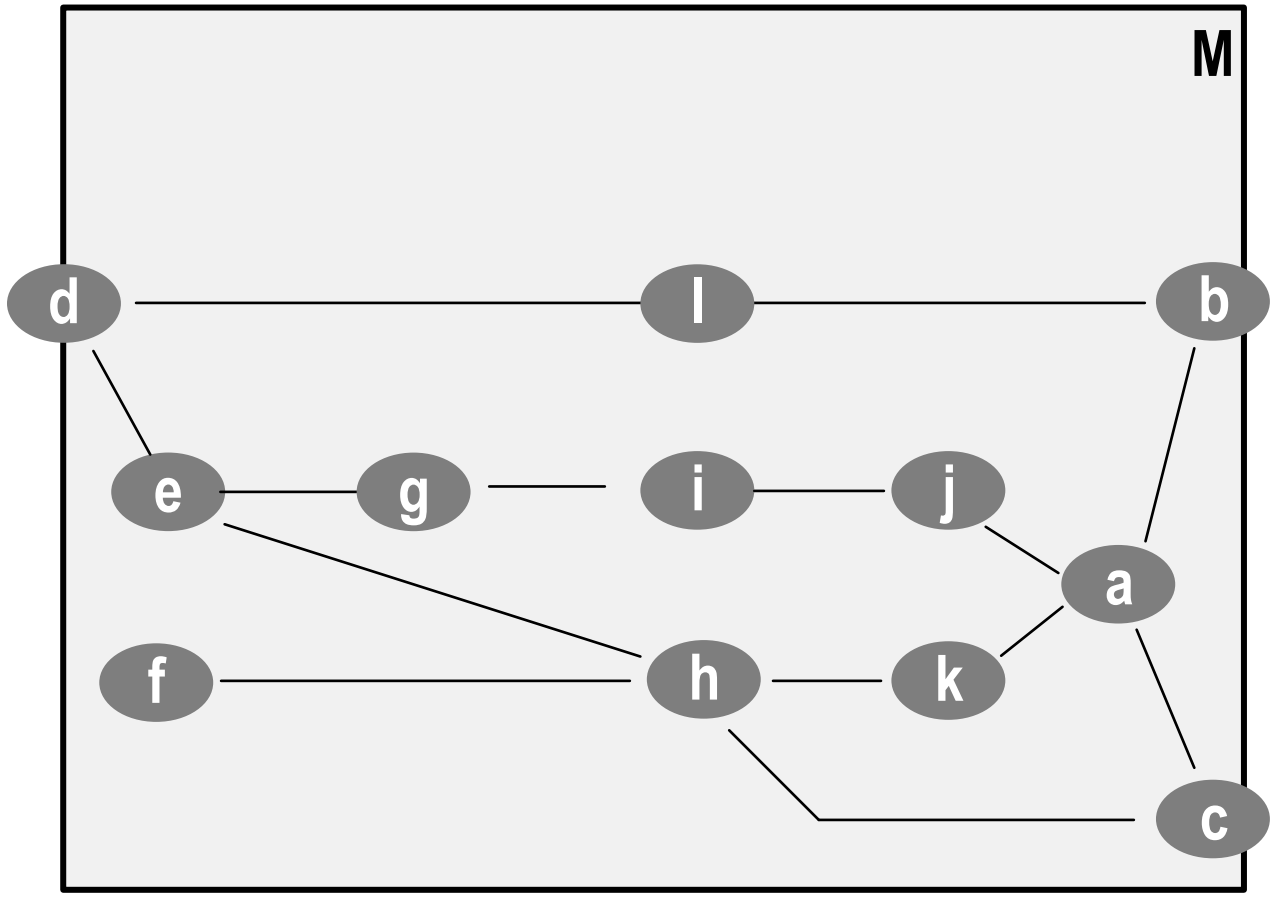
4. Distinguished interior
4.2 submodules

the complement modul of H



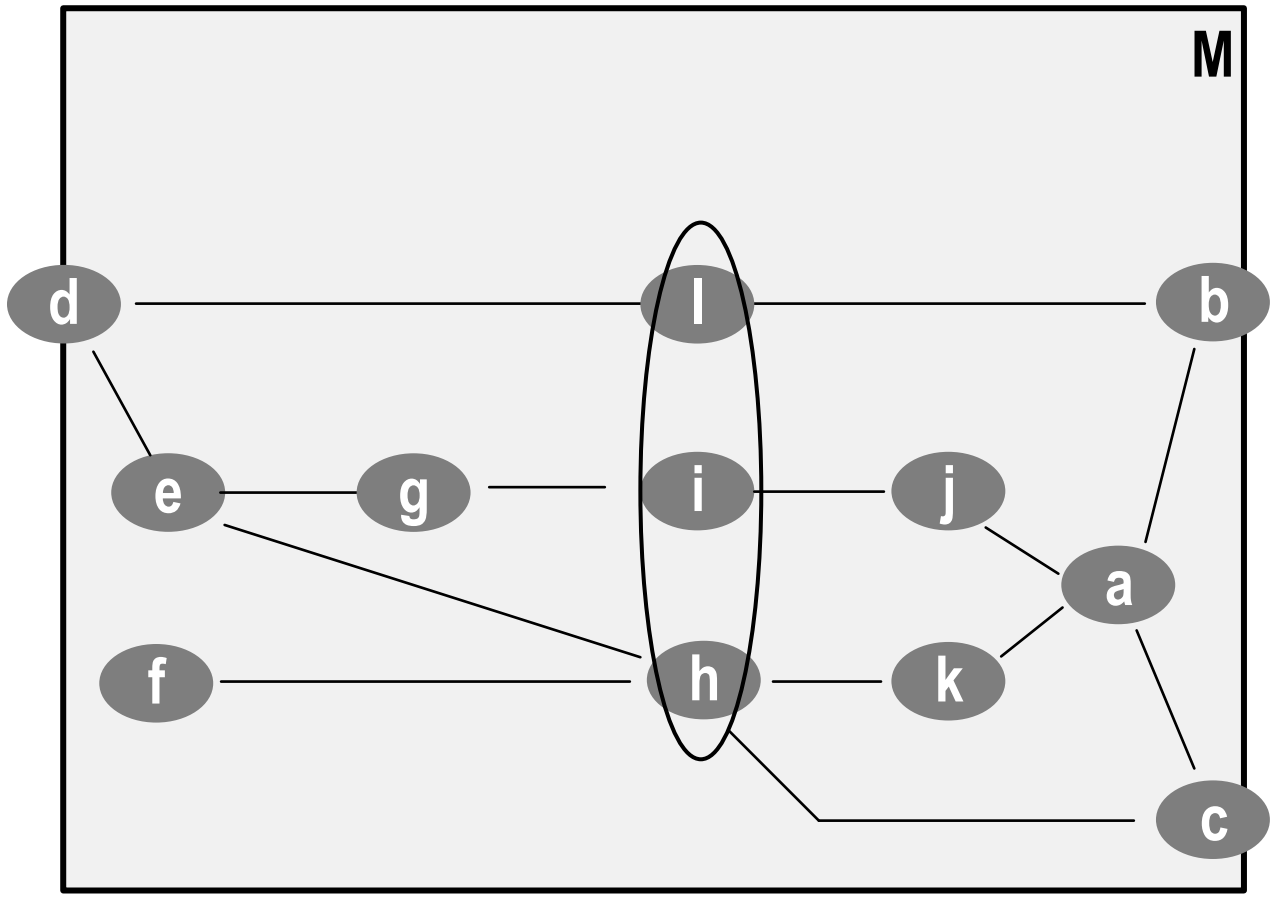
4. Distinguished interior
4.3 initial and final submodules

module M



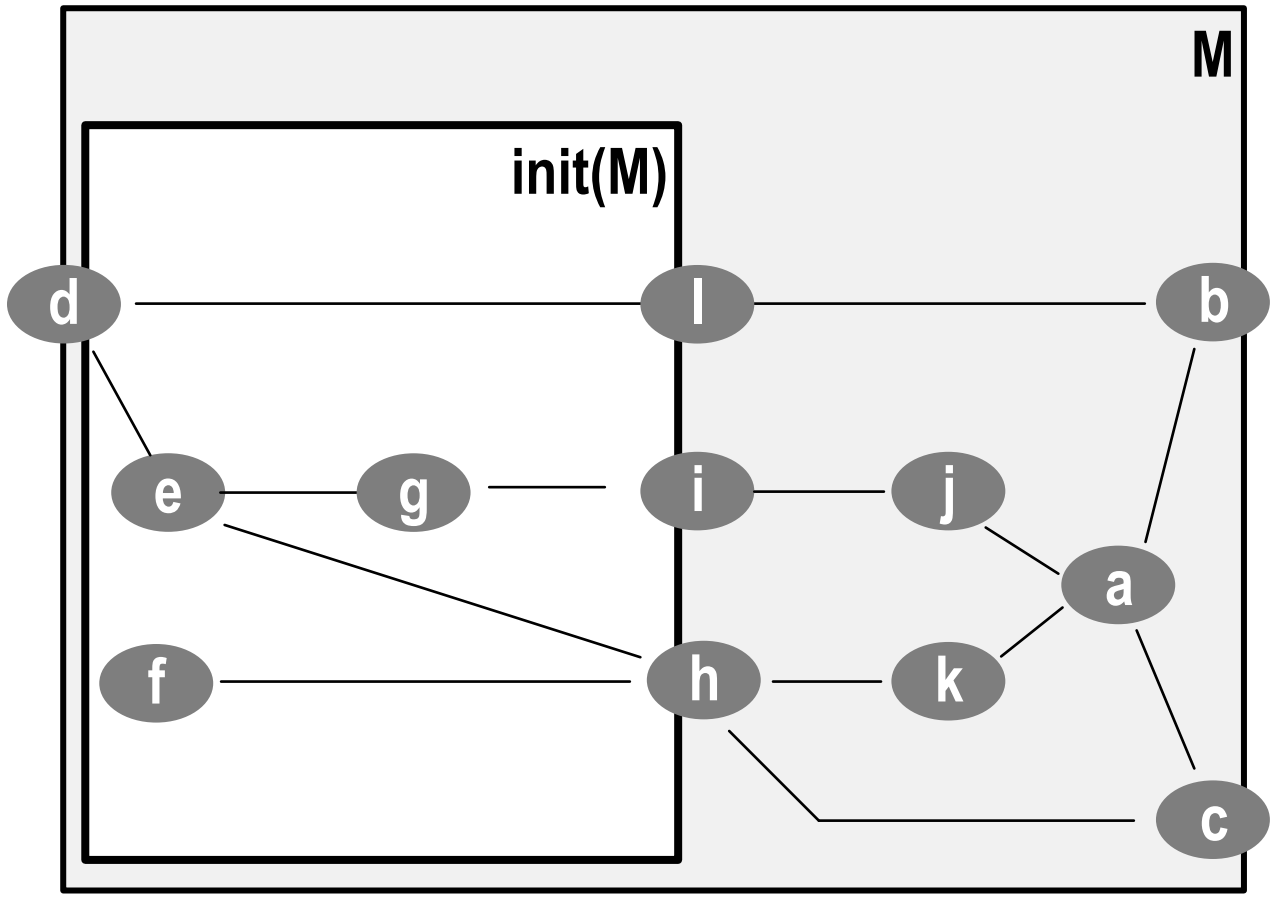
4. Distinguished interior
4.3 initial and final submodules

module M
a cut



4. Distinguished interior
4.3 initial and final submodules

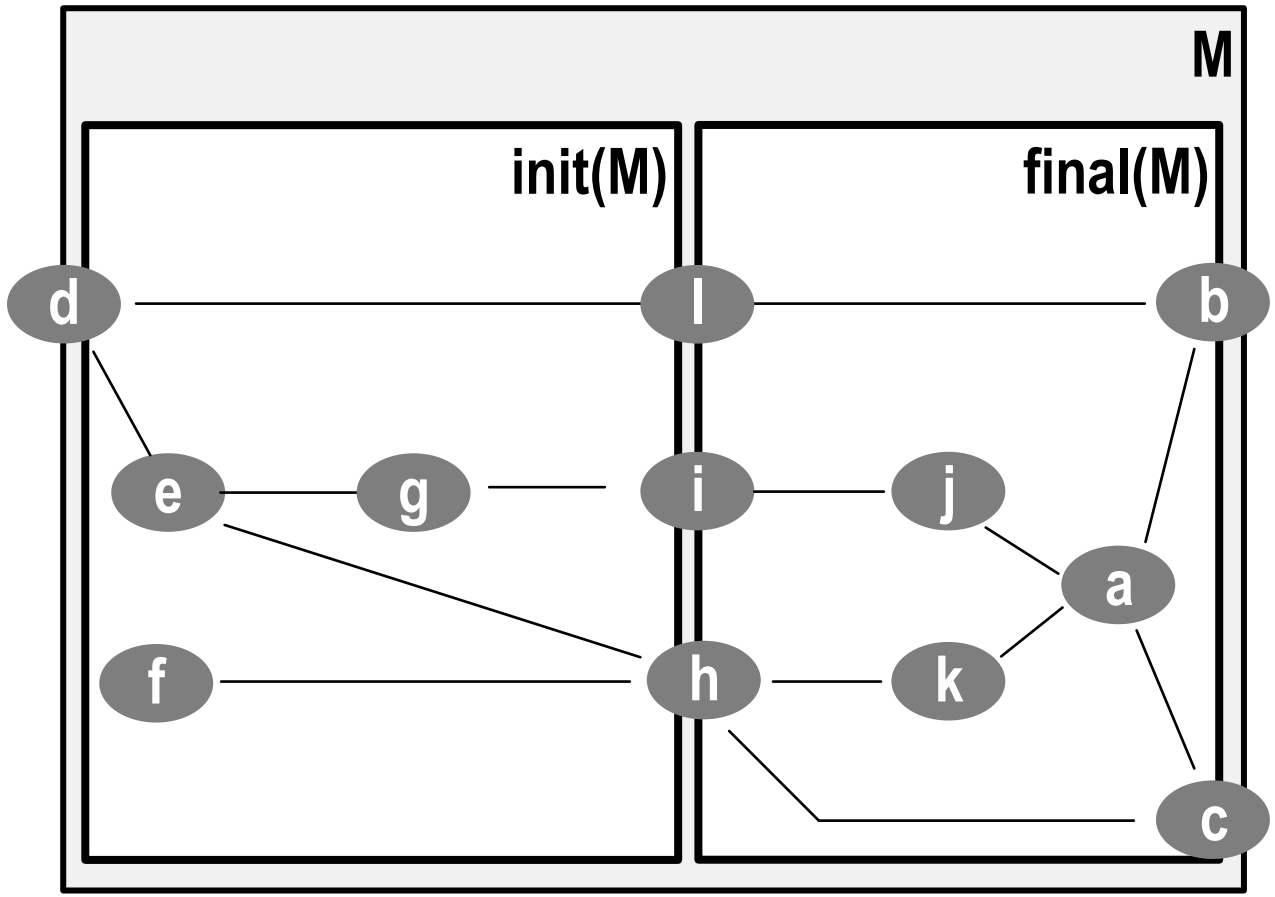
module M
a cut
an initial module



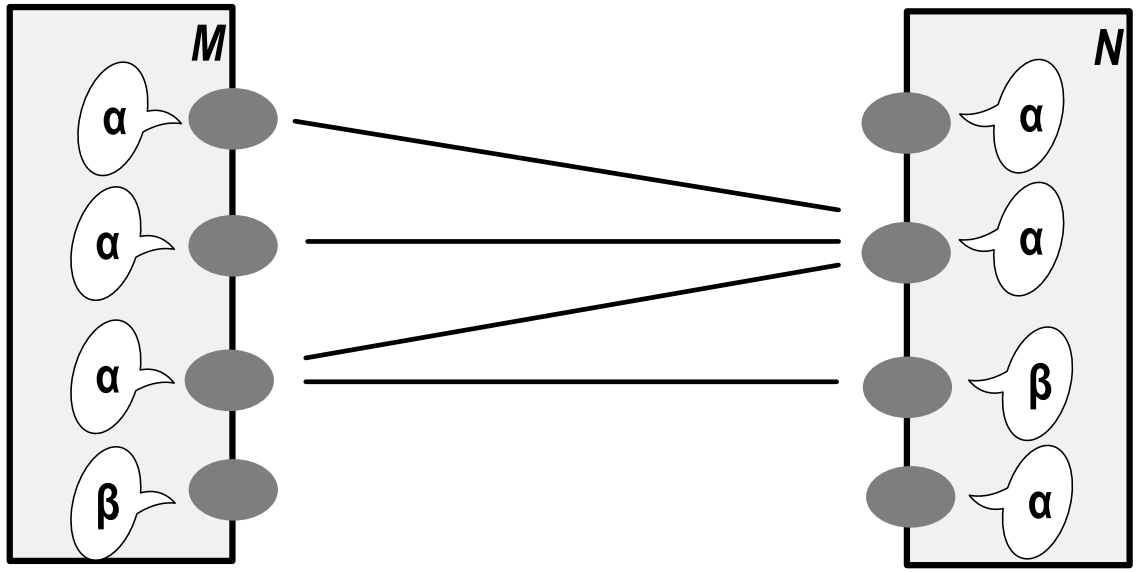
4. Distinguished interior
4.3 initial and final submodules

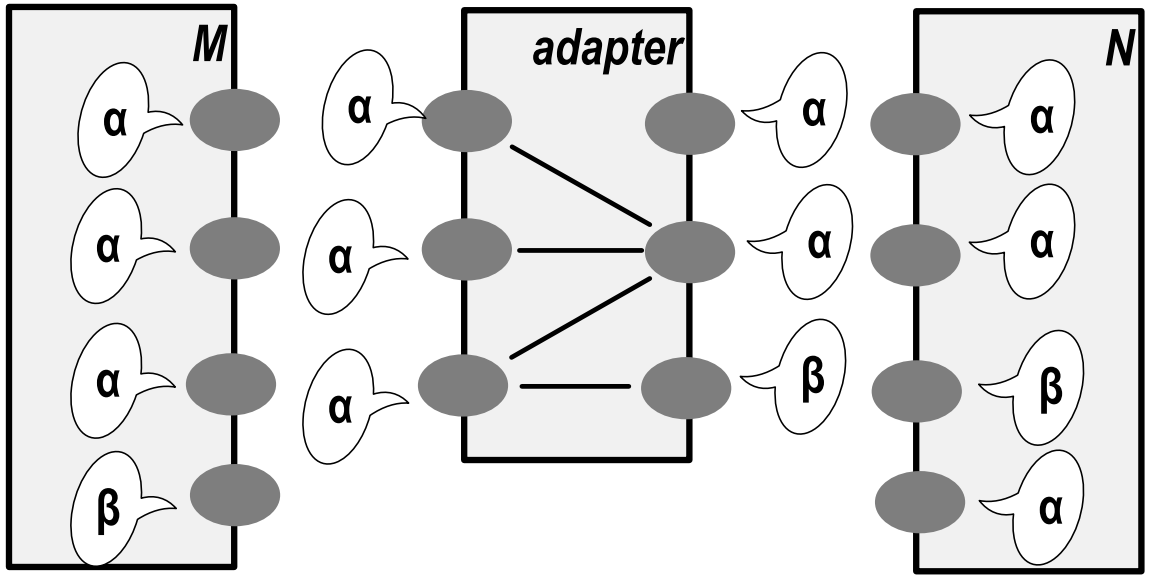
module M
a cut
an initial module
a final module

M =
init(M) • final(M)



4. Distinguished interior
4.4 Adapters





5. Miscellaneous

single modules
shared gates
perfect matches
atomic modules
abstract modules
reverse interface
hierarchies

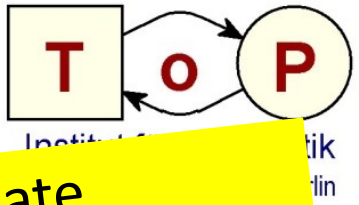
classes of modules
finitely generated classes
domain specific classes
Petri net modules

Once and for all: how to compose modules - the composition calculus

SummerSoc Thursday, June 27, 2024

- Composition calculus: not just another composition operator.
- A fundamental basis for any composition principle.
- There are more algebraic properties worth considering, in particular concerning hierarchies and submodules.
- Best practice concepts such as adapters and domain specific subcalculi are under consideration.

it's never too late to start using the compostion calculus!



UNIVERSITÄT DES SAARLANDES