Multirelational Semantics for Extended Entity–Relationship Schemata With Applications

Sebastiano Vigna

Dipartimento di Scienze dell'Informazione

Università degli Studi di Milano

Motivation

ERW

- A tool bridging conceptual modelling with web applications

ERW

- A tool bridging conceptual modelling with web applications
- Starting from any schema, generate a web-based content manager

ERW

- A tool bridging conceptual modelling with web applications
- Starting from any schema, generate a web-based content manager
- Wide range of EER schema features supported

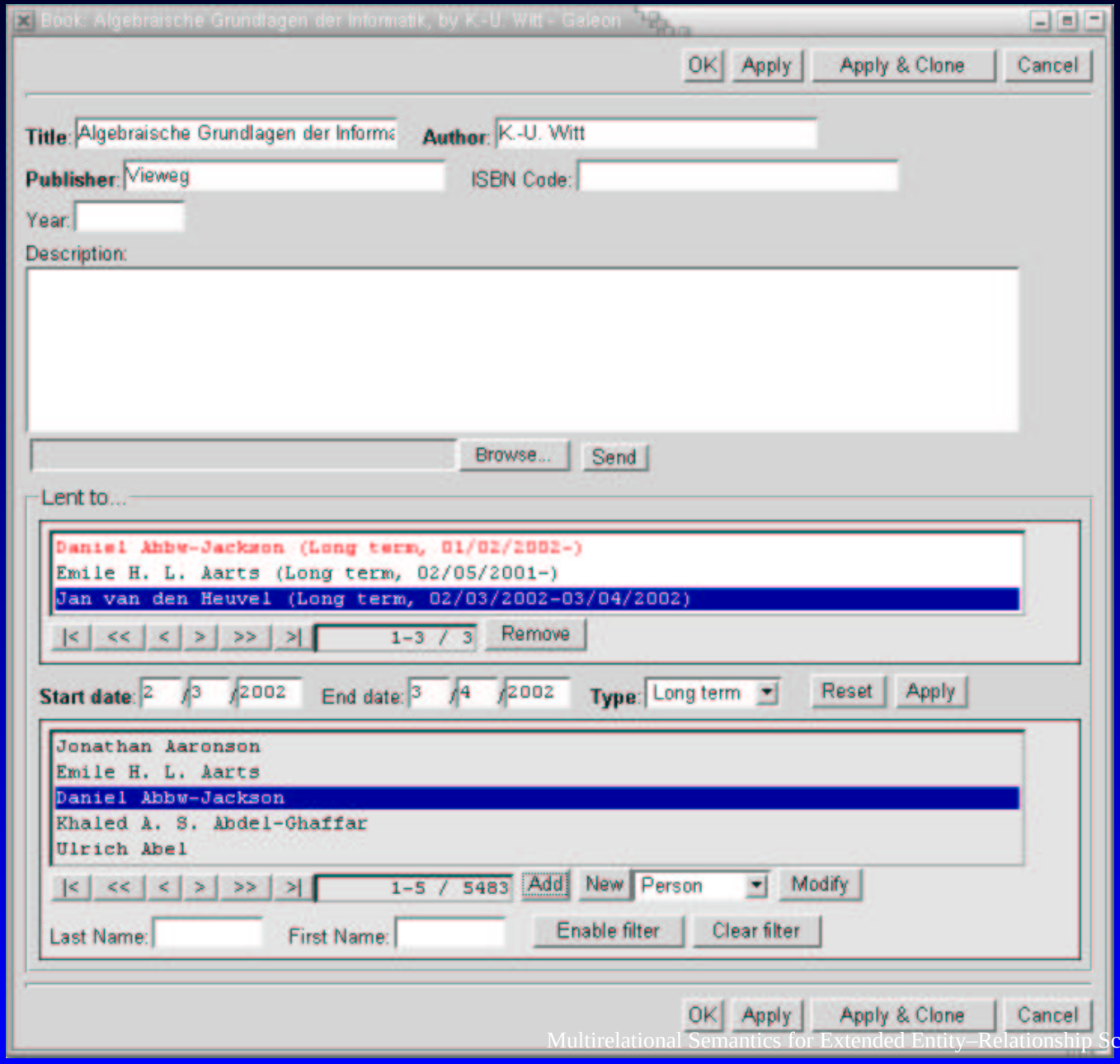
ERW

- A tool bridging conceptual modelling with web applications
- Starting from any schema, generate a web-based content manager
- Wide range of EER schema features supported
- Need for validation

ERW

- A tool bridging conceptual modelling with web applications
- Starting from any schema, generate a web-based content manager
- Wide range of EER schema features supported
- Need for validation
- Concept similar to WebML (Ceri, Fraternali & Bongio WWW 2000) but different scope and features

ERW


Book: Algebraische Grundlagen der Informatik, by K.-U. Witt - Galeon
OK
Apply
Apply & Clone
Cancel
Title: Algebraische Grundlagen der Informa
Author: K.-U. Witt
Publisher: Vieweg
ISBN Code:
Year:
Description:
Browse...
Send
Lent to...
Daniel Abbw-Jackson (Long term, 01/02/2002-)
Emile H. L. Aarts (Long term, 02/05/2001-)
Jan van den Heuvel (Long term, 02/03/2002-03/04/2002)
|< << < > >> >|
1-3 / 3
Remove
Start date: 2 /3 /2002
End date: 3 /4 /2002
Type: Long term
Reset
Apply
Jonathan Aaronson
Emile H. L. Aarts
Daniel Abbw-Jackson
Khaled A. S. Abdel-Ghaffar
Ulrich Abel
|< << < > >> >|
1-5 / 5483
Add
New
Person
Modify
Last Name:
First Name:
Enable filter
Clear filter
OK
Apply
Apply & Clone
Cancel

Validation

- *Acyclic typing*: type hierarchies must be DAGs (Directed Acyclic Graphs)

Validation

- *Acyclic typing*: type hierarchies must be DAGs (Directed Acyclic Graphs)
- *Full reachability*: all schema instances are mutually reachable by local transformations

Validation

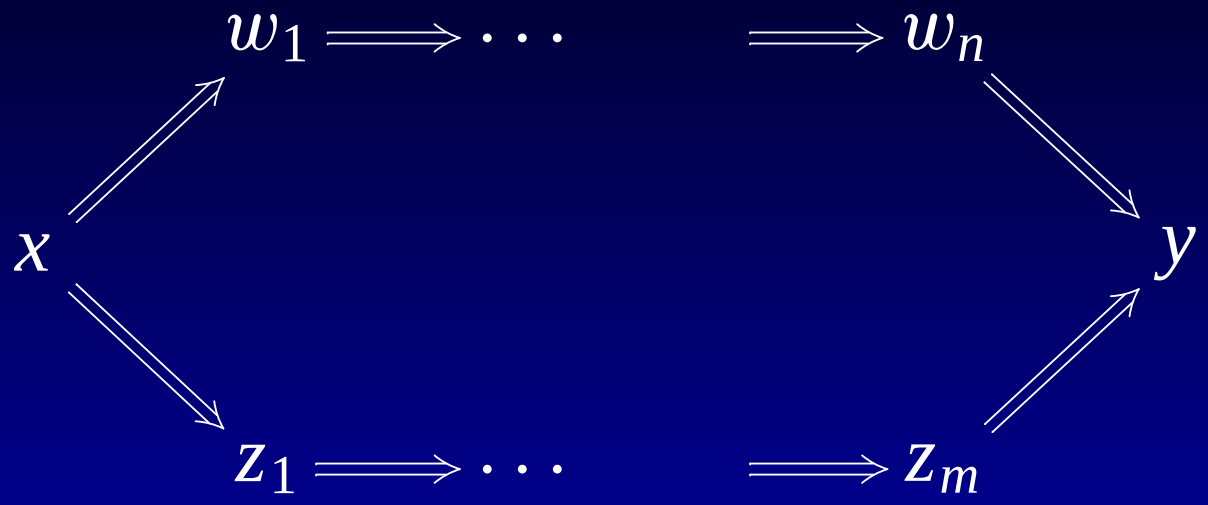
- *Acyclic typing*: type hierarchies must be DAGs (Directed Acyclic Graphs)
- *Full reachability*: all schema instances are mutually reachable by local transformations
- *No double ownership*: no schema instance contains two entities x and y such that y owns x along two distinct ownership paths

Validation

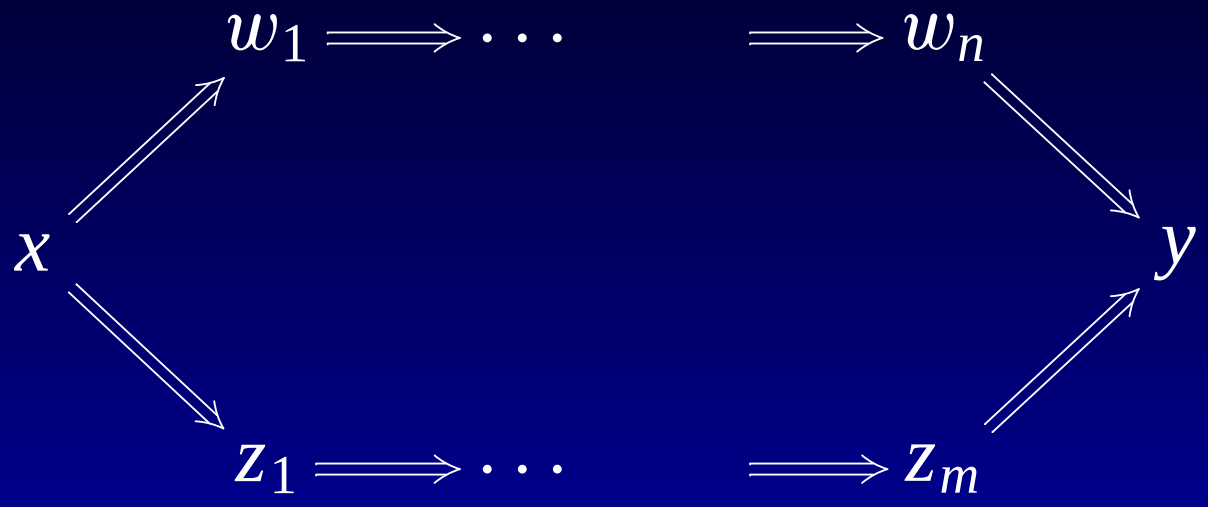
- *Acyclic typing*: type hierarchies must be DAGs (Directed Acyclic Graphs)
- *Full reachability*: all schema instances are mutually reachable by local transformations
- *No double ownership*: no schema instance contains two entities x and y such that y owns x along two distinct ownership paths
- We concentrate on the third condition (the "application" in the title)

Double Ownership

Double Ownership

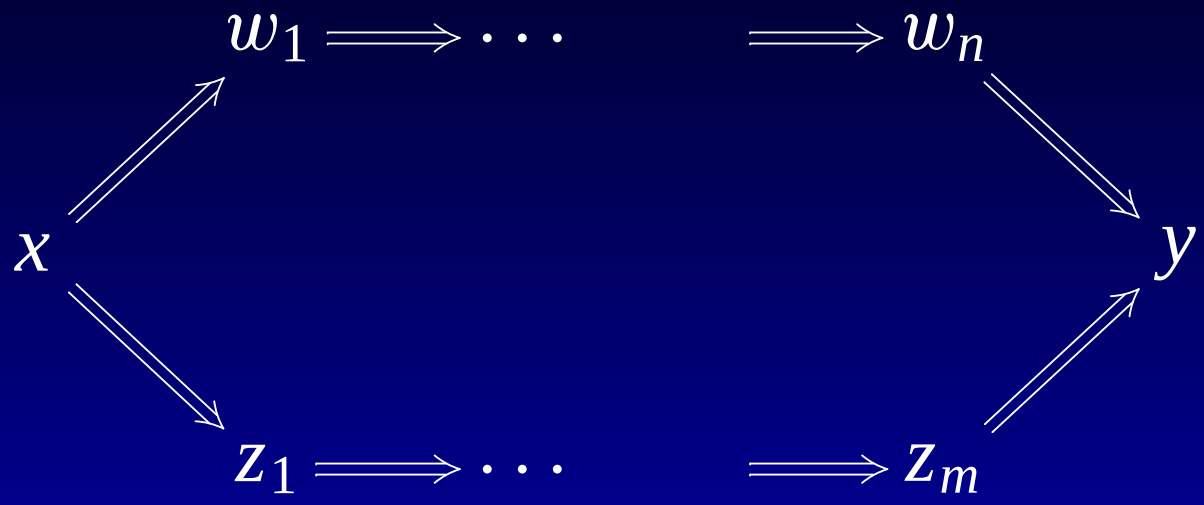


Double Ownership

- 

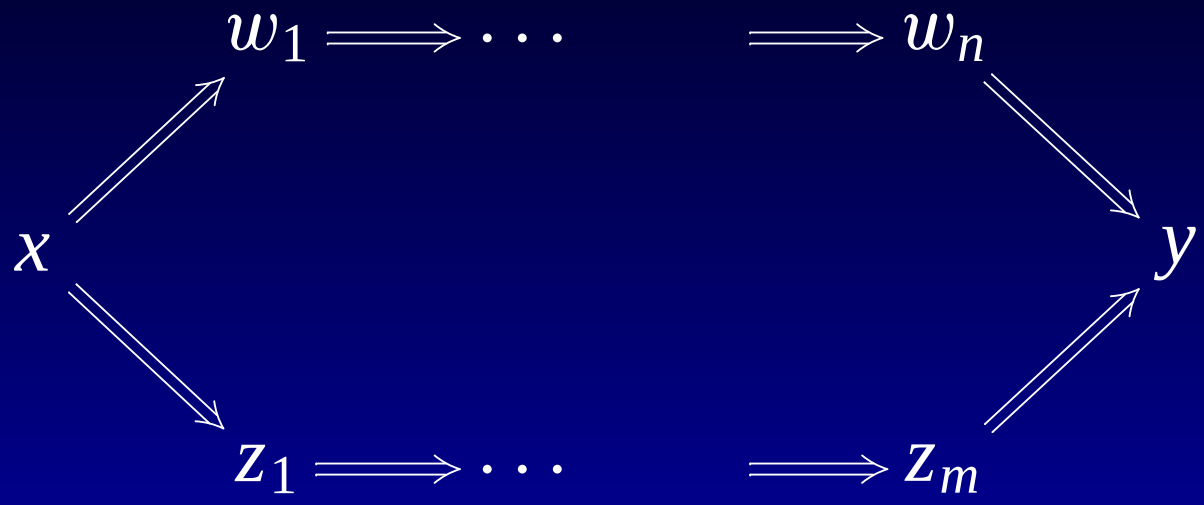
- It is a *semantic* condition: it happens in a schema *instance*

Double Ownership



- It is a *semantic* condition: it happens in a schema *instance*
- We are interested in rejecting *syntactically* the bad schemata

Double Ownership



- It is a *semantic* condition: it happens in a schema *instance*
- We are interested in rejecting *syntactically* the bad schemata
- You need a formal semantics, or things mess up (e.g., Balaban & Shoval ER 1999)

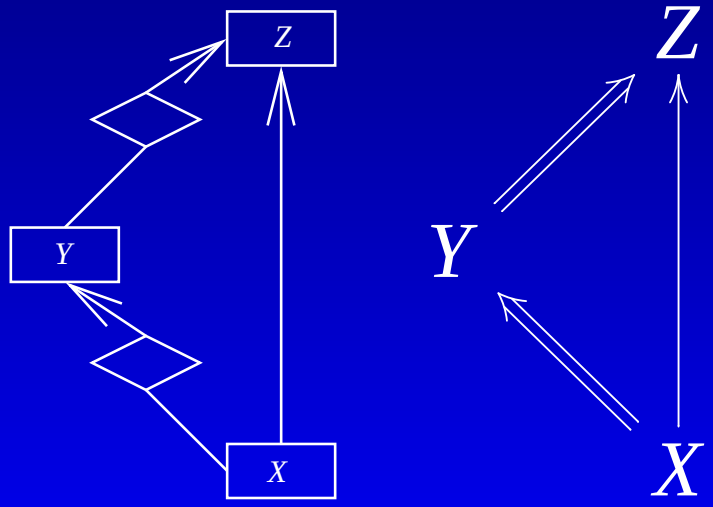
The Typeless Case

- Just require that no pair of parallel paths (same source and destination) of identification functions exist (e.g., Thalheim's book)

The Typeless Case

- Just require that no pair of parallel paths (same source and destination) of identification functions exist (e.g., Thalheim's book)
- Does *not* work with types (a.k.a. "is-a", generalisation, specialisation, etc.)

The Typeless Case

- Just require that no pair of parallel paths (same source and destination) of identification functions exist (e.g., Thalheim's book)
- Does *not* work with types (a.k.a. "is-a", generalisation, specialisation, etc.)
- 

Kluges

- Consider subtyping a form of ownership

Kluges

- Consider subtyping a form of ownership
- Collapse type hierarchies into points, and check for parallel paths

Kluges

- Consider subtyping a form of ownership
- Collapse type hierarchies into points, and check for parallel paths
- Use only minimal types

Kluges

- Consider subtyping a form of ownership
- Collapse type hierarchies into points, and check for parallel paths
- Use only minimal types
- . . . but none of the above works

Kluges

- Consider subtyping a form of ownership
- Collapse type hierarchies into points, and check for parallel paths
- Use only minimal types
- . . . but none of the above works
- To prove a soundness and completeness result we need a formal, mathematical semantics (e.g., see Gogolla & Hohenstein 1991, Thalheim 2000,. . .)

Semantics

Semantics

- Literature widely disagrees

Semantics

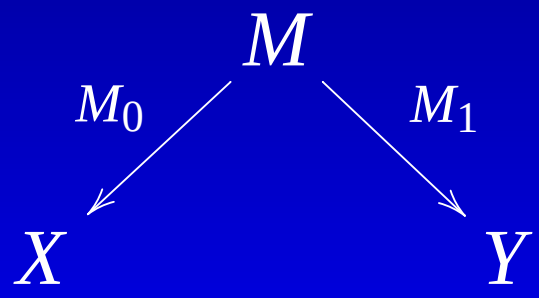
- Literature widely disagrees
- Semantics of ER schemata should not be defined in term of tuples, but rather in terms of sets and elements (Chen 1976)

Semantics

- Literature widely disagrees
- Semantics of ER schemata should not be defined in term of tuples, but rather in terms of sets and elements (Chen 1976)
- At this level of abstraction, you need multirelations

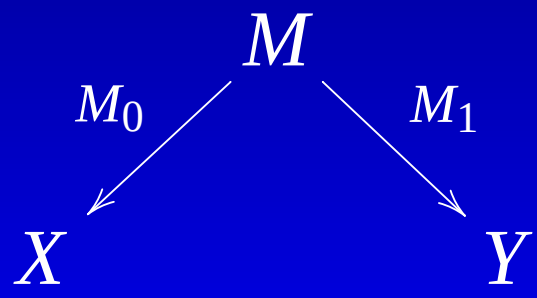
Semantics

- Literature widely disagrees
- Semantics of ER schemata should not be defined in term of tuples, but rather in terms of sets and elements (Chen 1976)
- At this level of abstraction, you need multirelations
- Easy formulation from bicategory theory:



Semantics

- Literature widely disagrees
- Semantics of ER schemata should not be defined in term of tuples, but rather in terms of sets and elements (Chen 1976)
- At this level of abstraction, you need multirelations
- Easy formulation from bicategory theory:



- Provides ready-made composition etc.

Types

- What does it mean to say that X "is a" Y (X is a subtype of Y)?

Types

- What does it mean to say that X "is a" Y (X is a subtype of Y)?
- The usual answer: in every instance, the set giving semantics to X is a *subset* of the one giving semantics to Y

Types

- What does it mean to say that X "is a" Y (X is a subtype of Y)?
- The usual answer: in every instance, the set giving semantics to X is a *subset* of the one giving semantics to Y
- Said otherwise, every entity of type X is also an entity of type Y

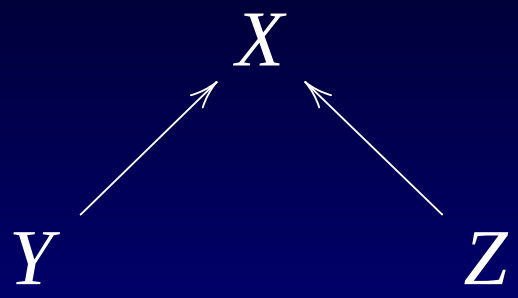
Types

- What does it mean to say that X "is a" Y (X is a subtype of Y)?
- The usual answer: in every instance, the set giving semantics to X is a *subset* of the one giving semantics to Y
- Said otherwise, every entity of type X is also an entity of type Y
- Has every entity a definite, unique type (as in programming languages)?

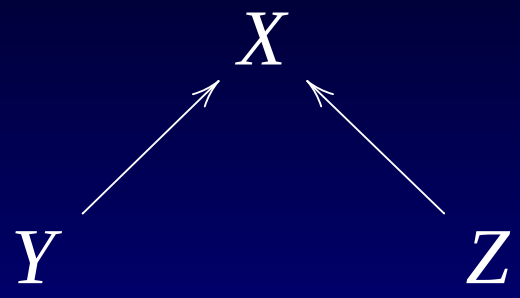
Types

- What does it mean to say that X "is a" Y (X is a subtype of Y)?
- The usual answer: in every instance, the set giving semantics to X is a *subset* of the one giving semantics to Y
- Said otherwise, every entity of type X is also an entity of type Y
- Has every entity a definite, unique type (as in programming languages)?
- Not in general!

Types (2)

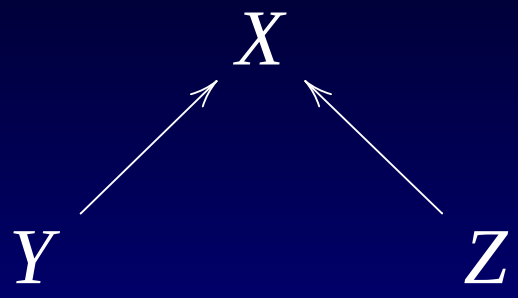


Types (2)



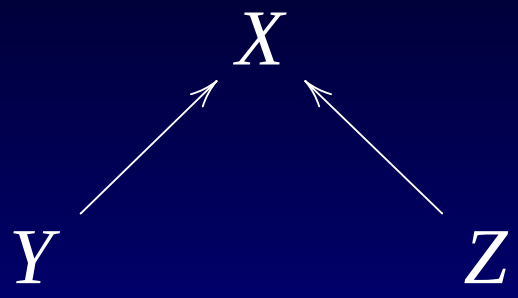
- What if w is of type X, Y *and* Z?

Types (2)



- What if w is of type X, Y *and* Z?
- This satisfies the condition above, but give rise to entities without a definite type (as in programming languages and type systems)

Types (2)



- What if w is of type X, Y *and* Z?
- This satisfies the condition above, but give rise to entities without a definite type (as in programming languages and type systems)
- Conjunctive vs. disjunctive subtyping

Conjunctive (or Free) Typing

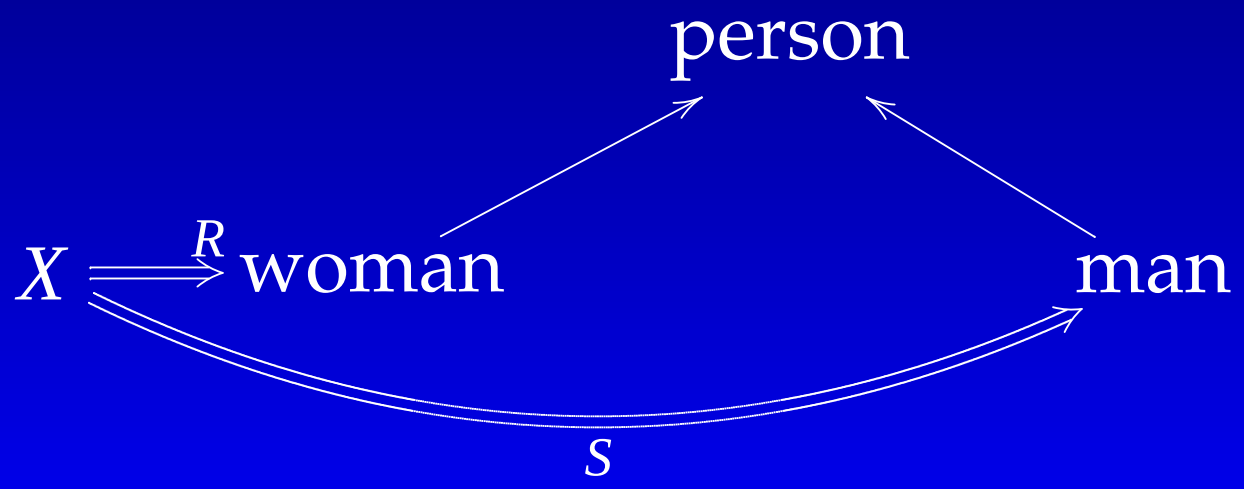
- If you accept this view, it is easy to show that once you collapse type hierarchies (i.e., shrink the weakly connected components of the type graph to points) Thalheim's test will work

Conjunctive (or Free) Typing

- If you accept this view, it is easy to show that once you collapse type hierarchies (i.e., shrink the weakly connected components of the type graph to points) Thalheim's test will work
- But is this what we need?

Conjunctive (or Free) Typing

- If you accept this view, it is easy to show that once you collapse type hierarchies (i.e., shrink the weakly connected components of the type graph to points) Thalheim's test will work
- But is this what we need?
-

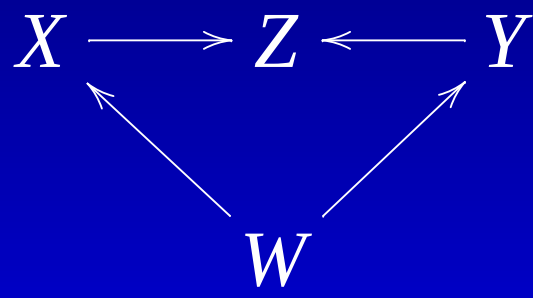


Definite Typing

- Type semantics should be constrained so that all entities have a definite type

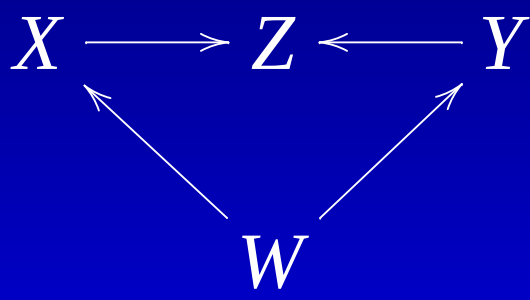
Definite Typing

- Type semantics should be constrained so that all entities have a definite type
- Intuitively, if entity w is both of type X *and* of type Y, and there is a supertype Z of both X and Y, then there must be a common subtype W such that w is also of type W



Definite Typing

- Type semantics should be constrained so that all entities have a definite type
- Intuitively, if entity w is both of type X *and* of type Y, and there is a supertype Z of both X and Y, then there must be a common subtype W such that w is also of type W



- Mathematically, you need a *stability* condition on the semantic map (Berry 1978, see the paper)

Definite Typing (2)

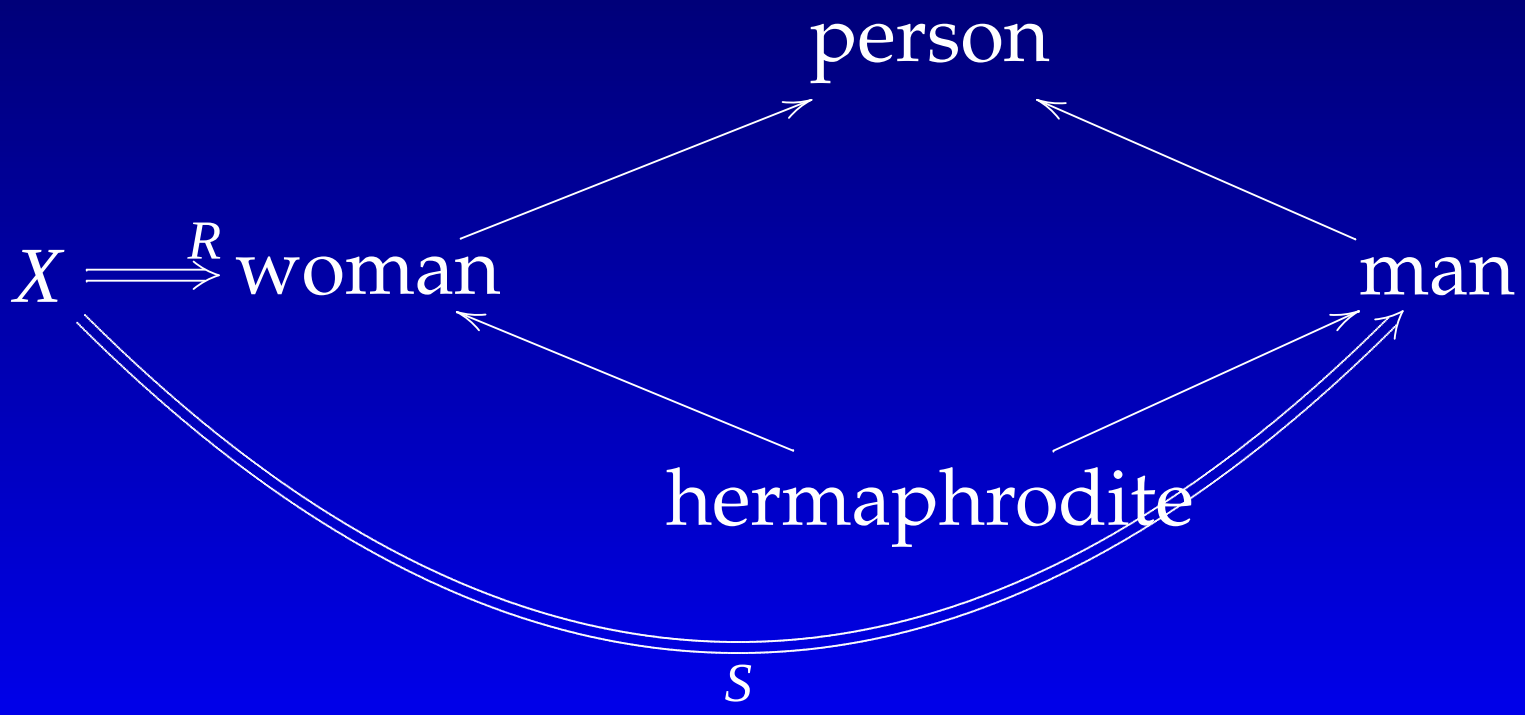
- You do *not* lose expressiveness!

Definite Typing (2)

- You do *not* lose expressiveness!
- If you need non-disjunctive subtyping, *just add explicitly the types you need*

Definite Typing (2)

- You do *not* lose expressiveness!
- If you need non-disjunctive subtyping, *just add explicitly the types you need*
-



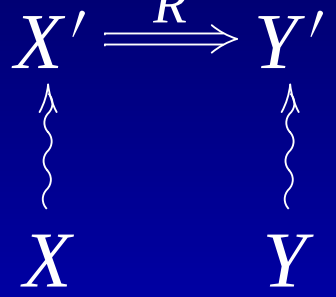
Validating Ownership

Syntax vs. Semantics

- The fact that *y* owns *x* has a syntactic counterpart

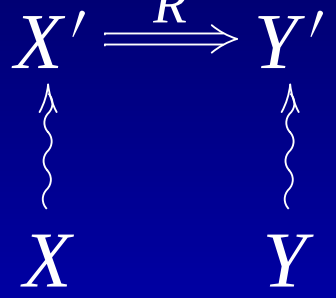
Syntax vs. Semantics

- The fact that y owns x has a syntactic counterpart
- Inheritance gives rise to *extended identification functions*:



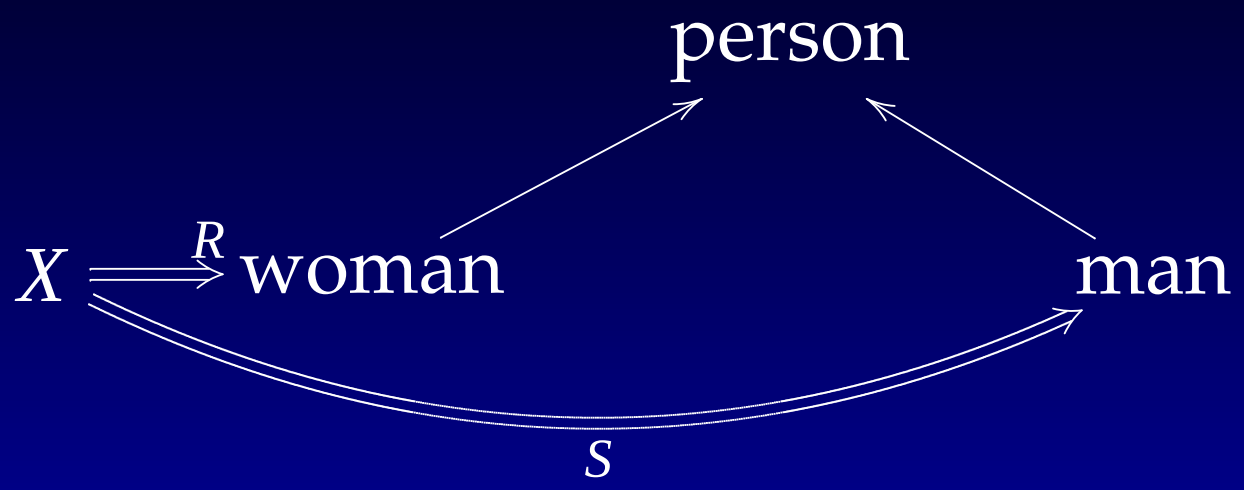
Syntax vs. Semantics

- The fact that y owns x has a syntactic counterpart
- Inheritance gives rise to *extended identification functions*:

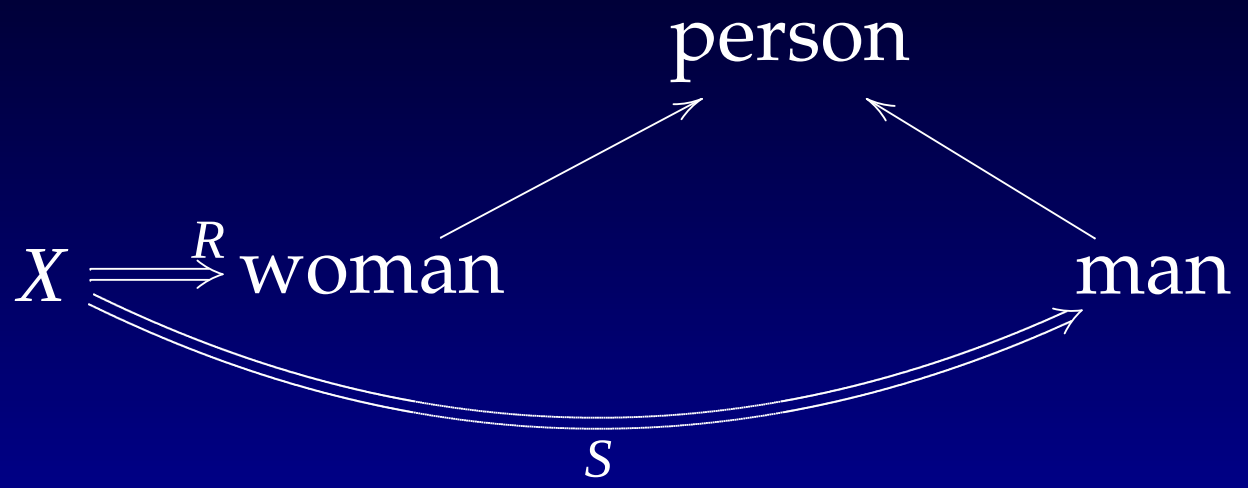


- Can we just build a new graph with extended identification functions, and check that no pair of parallel paths exists?

It May Work!

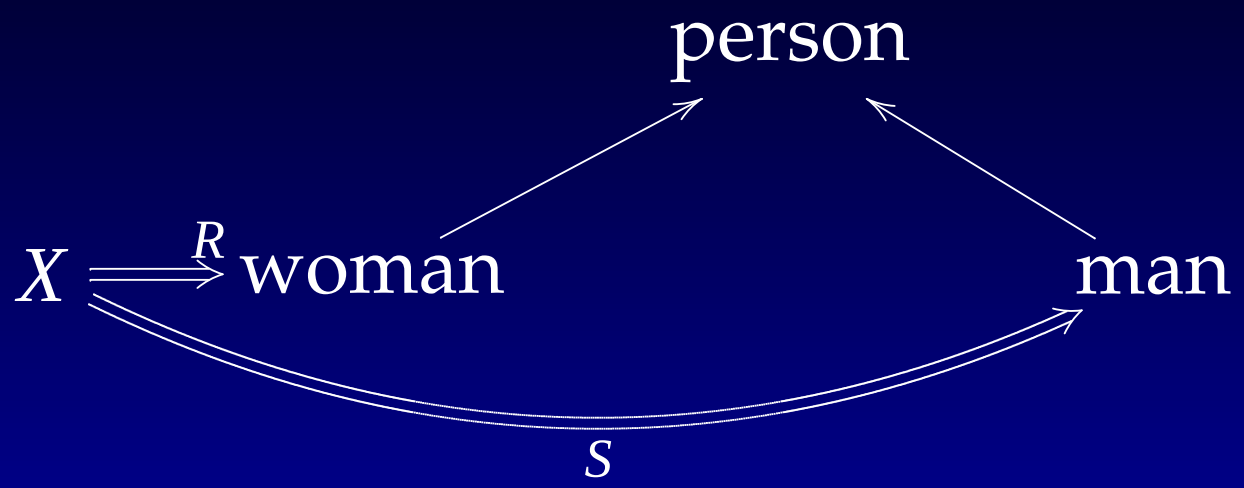


It May Work!

- 

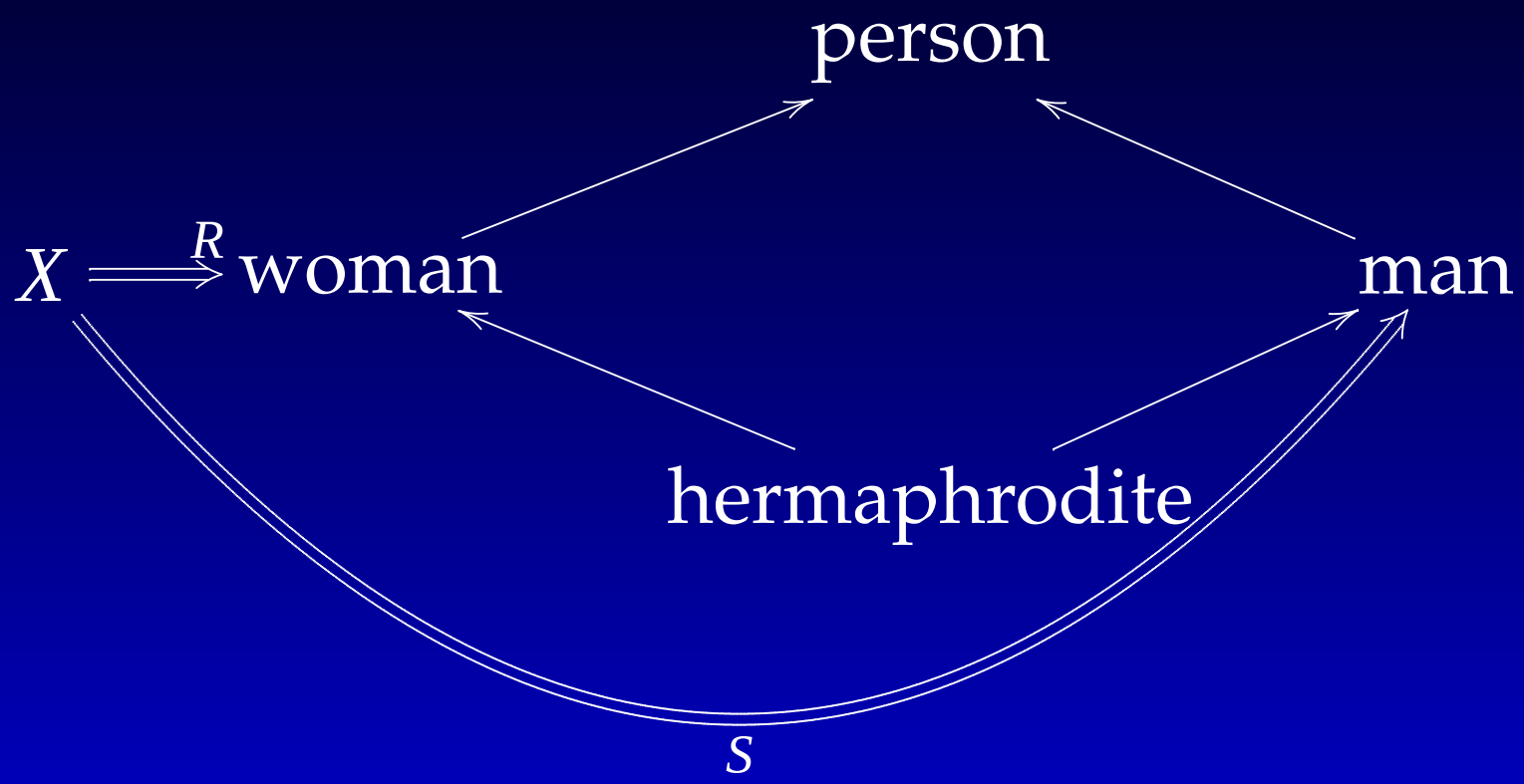
- No additional arcs

It May Work!

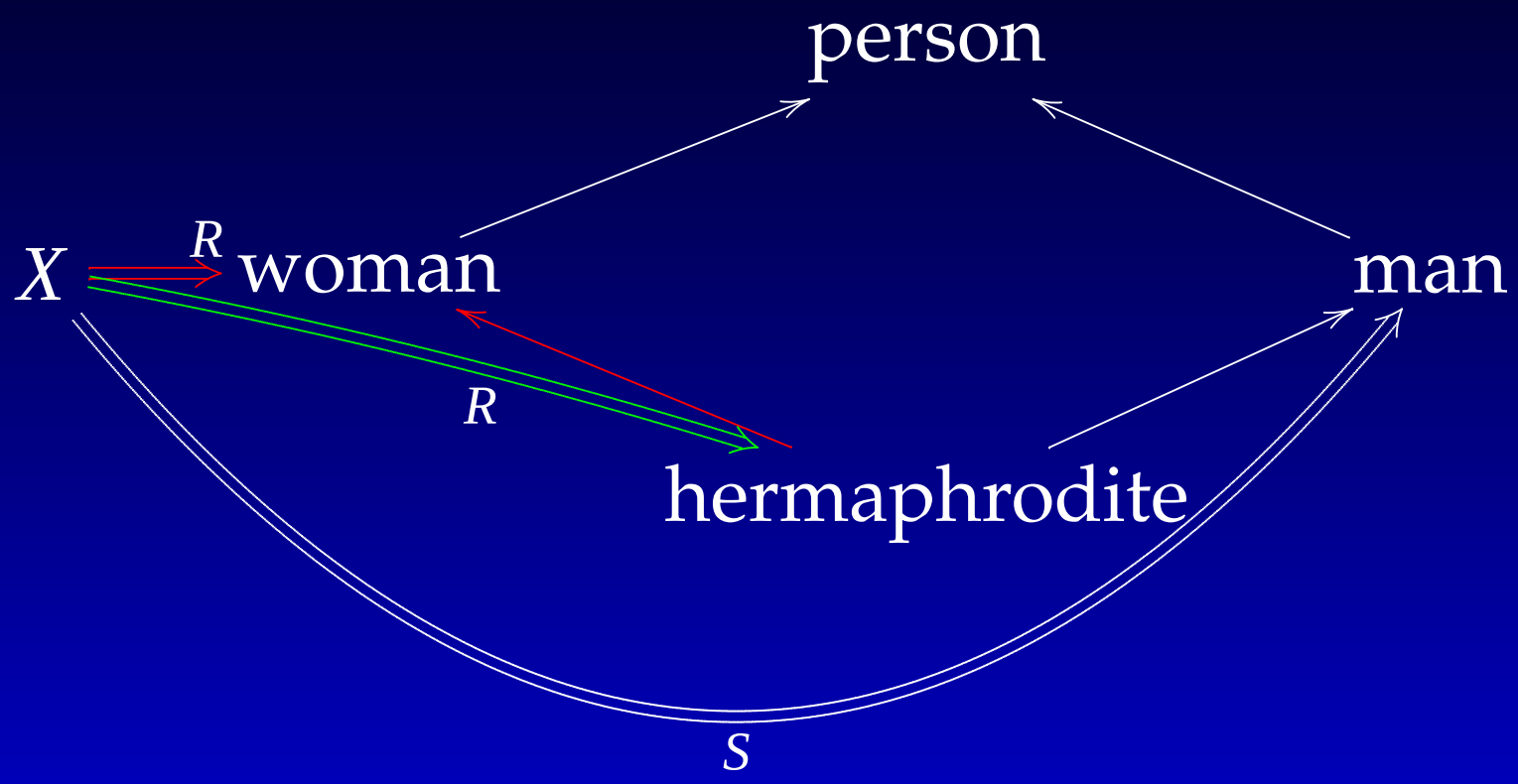


- No additional arcs
- The schema is valid

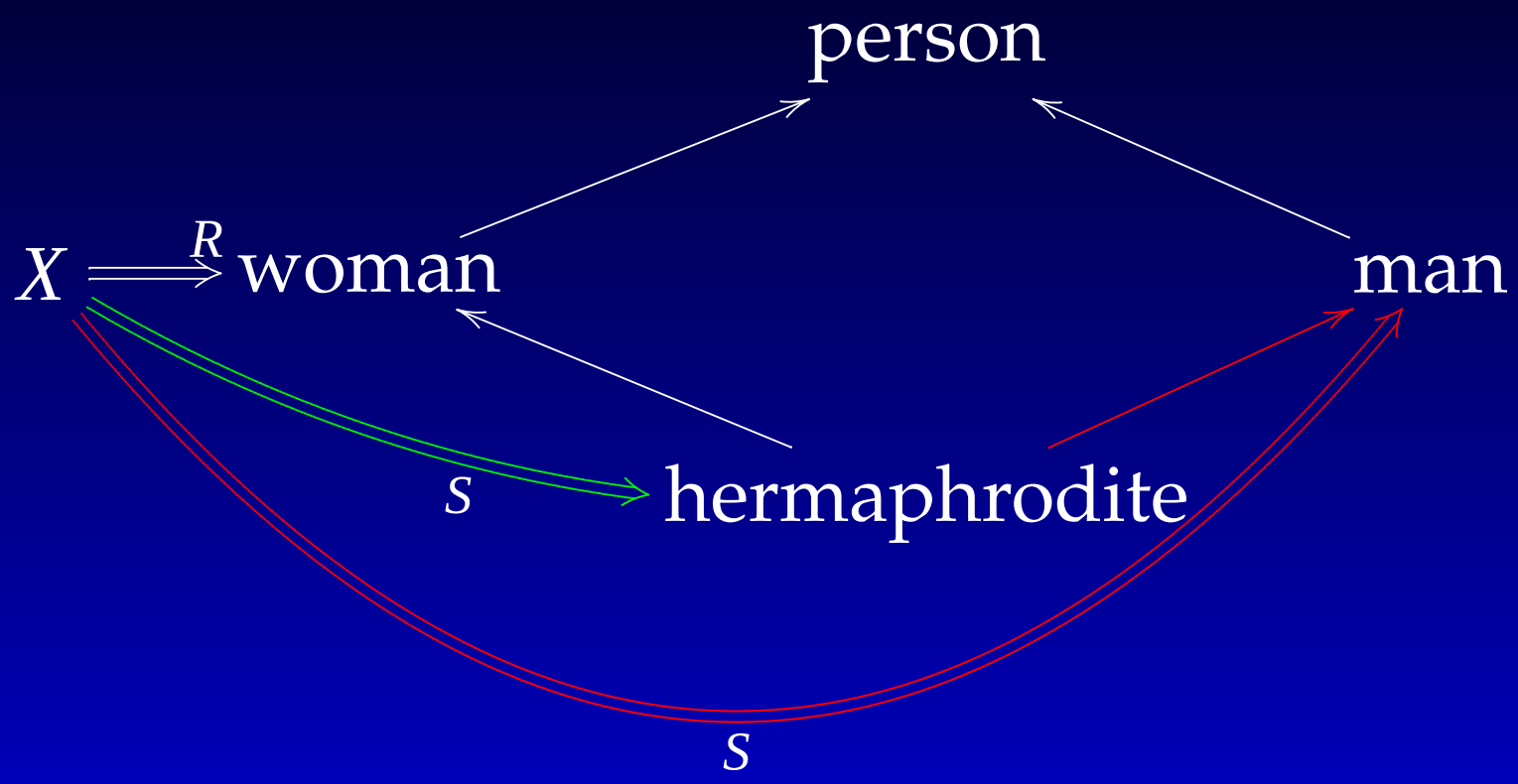
It May Work!



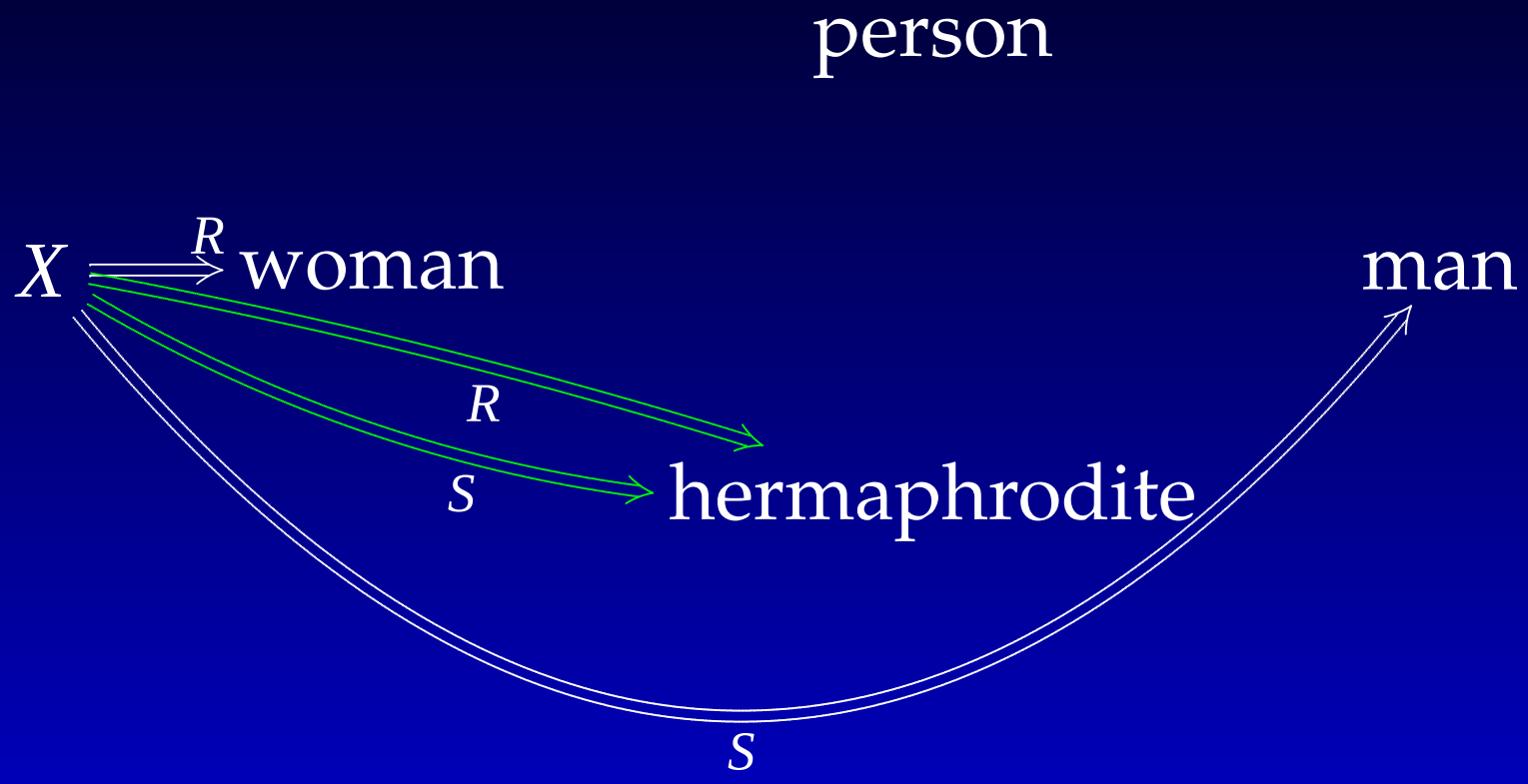
It May Work!



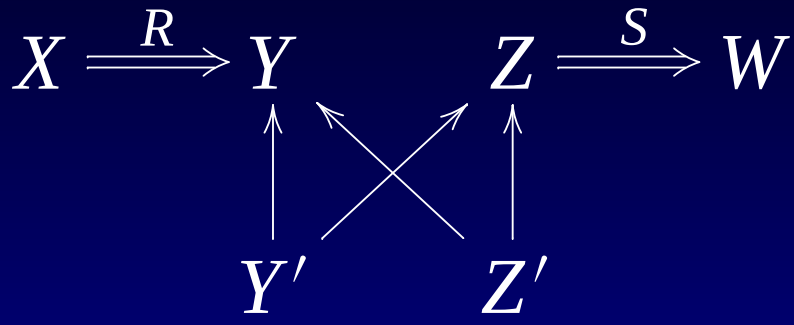
It May Work!



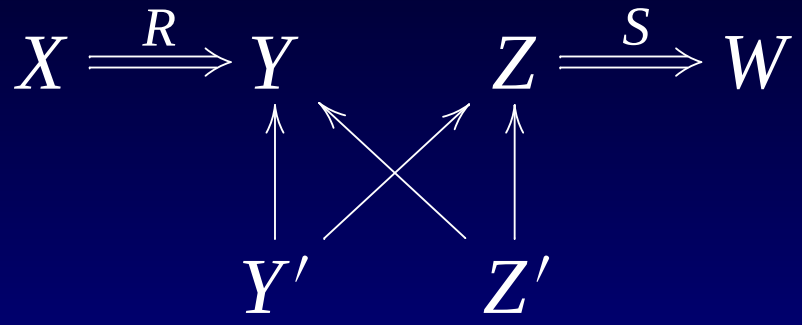
It May Work!

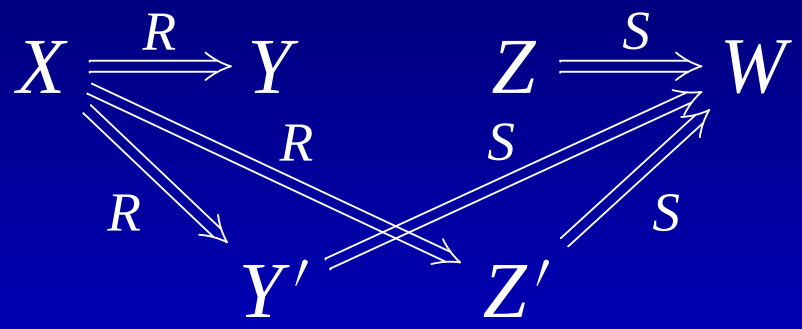


But it really doesn't...

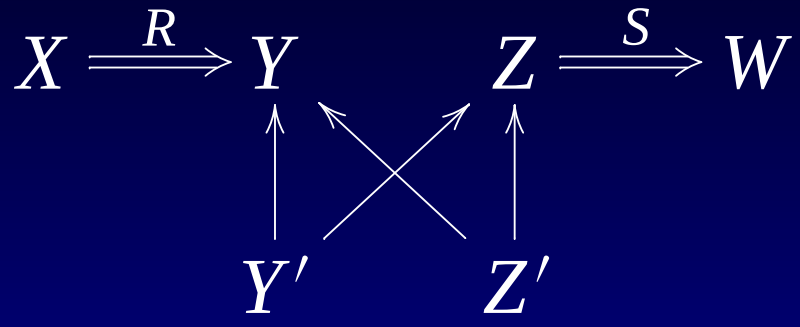


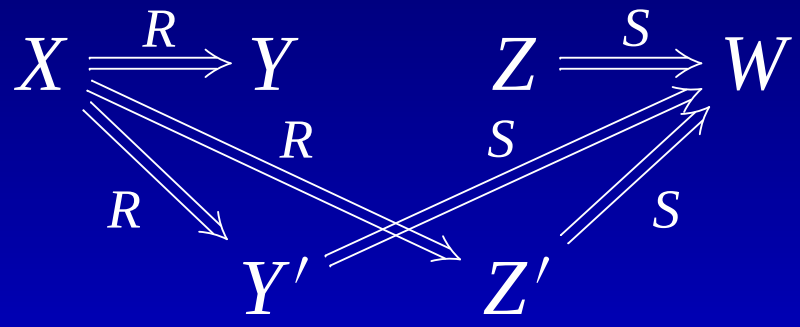
But it really doesn't...

- 

- 

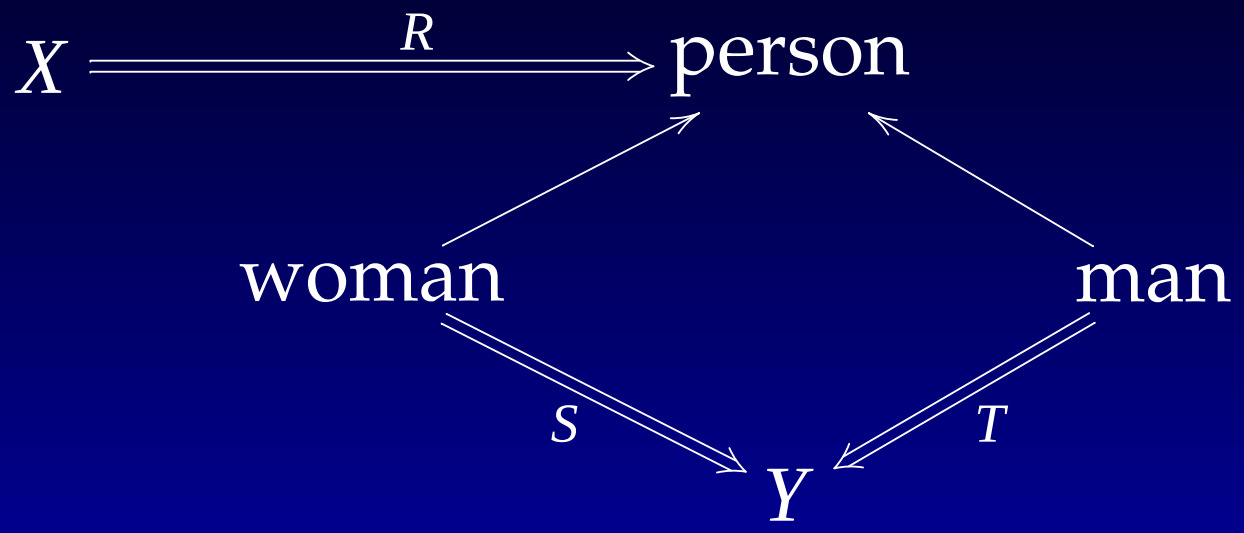
But it really doesn't...

- 

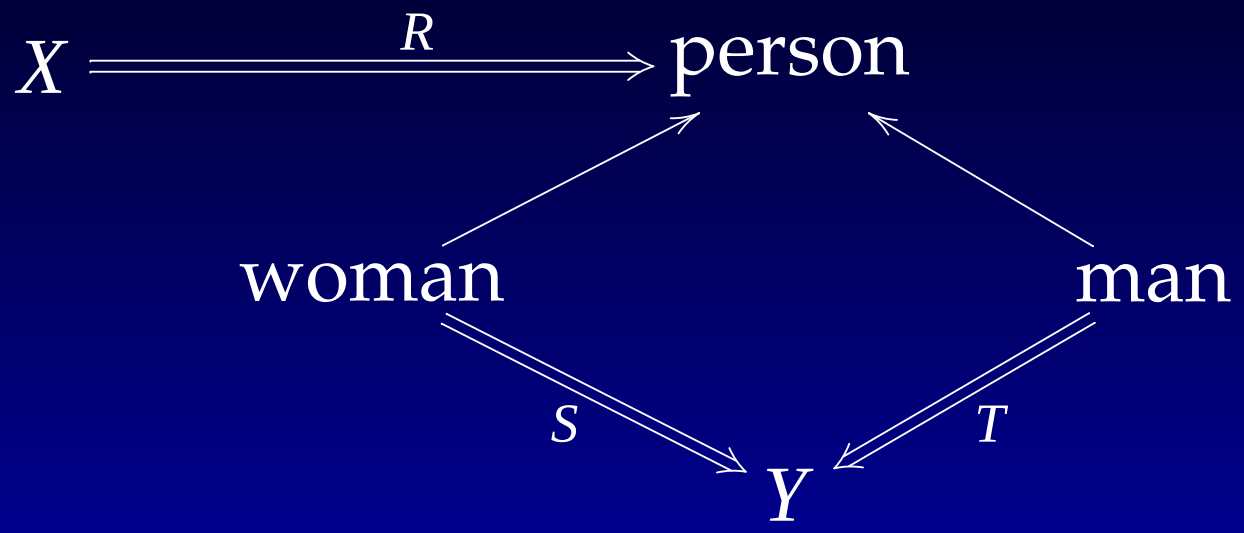
- 

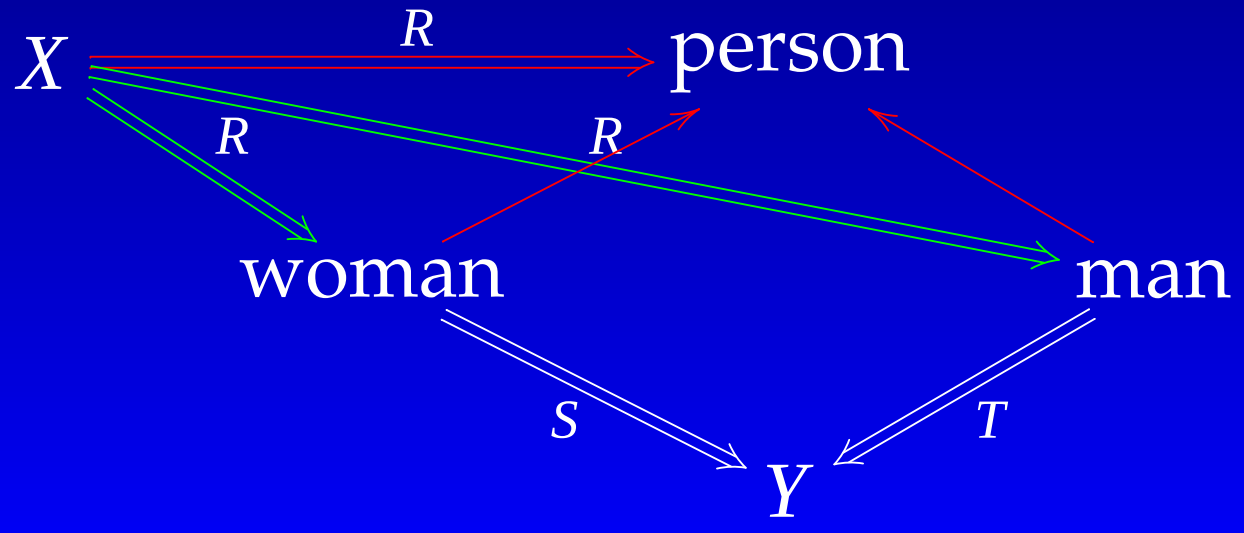
- Thus, not all pairs of parallel paths are bad. Maybe we should check that the sequence of labels (identification functions) is different...

. . . and Maybe Not



. . . and Maybe Not



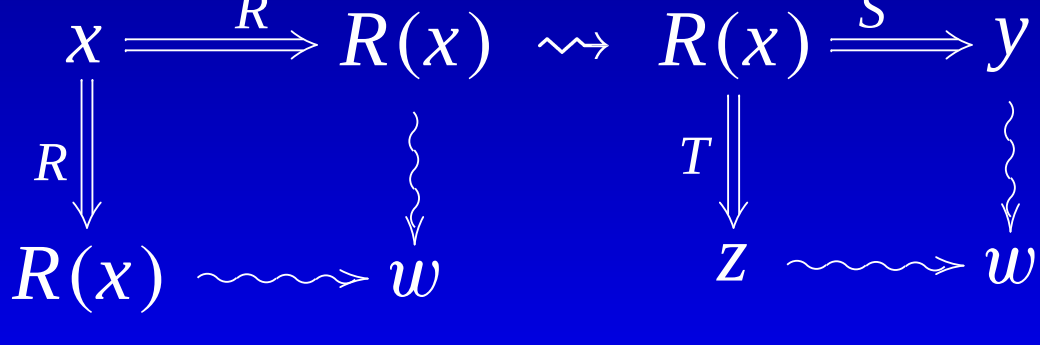


The characterisation

- The main result: double ownership can happen *if and only if* there is a *nonempty cycle* or there are two parallel paths *whose first label is different*.

The characterisation

- The main result: double ownership can happen *if and only if* there is a *nonempty cycle* or there are two parallel paths *whose first label is different*.
- Indeed, if double ownership is possible, then it must happen in such a way that *the first identification function is different*:



A Polynomial Algorithm

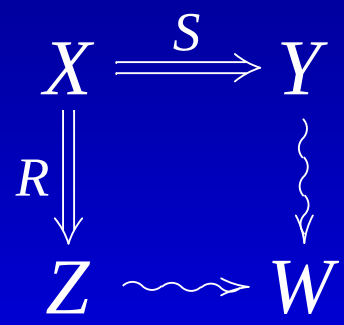
- Build the graph of extended identification functions

A Polynomial Algorithm

- Build the graph of extended identification functions
- Compute reachability for all nodes

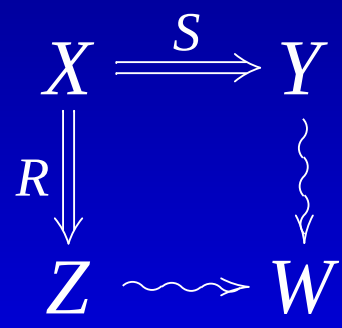
A Polynomial Algorithm

- Build the graph of extended identification functions
- Compute reachability for all nodes
- Take pairs of arcs with common source and distinct labels, and check whether their targets reach a common node; in this case, reject



A Polynomial Algorithm

- Build the graph of extended identification functions
- Compute reachability for all nodes
- Take pairs of arcs with common source and distinct labels, and check whether their targets reach a common node; in this case, reject



- Everything here is polynomial (albeit of rather high degree)

Faster, Faster, Faster,...

- The algorithm above is redundant

Faster, Faster, Faster,...

- The algorithm above is redundant
- If there are several ways to move between two types, we consider them all

Faster, Faster, Faster,...

- The algorithm above is redundant
- If there are several ways to move between two types, we consider them all
- We can do better taking an identification function followed by subtyping and supertyping (in this order)

Faster, Faster, Faster,...

- The algorithm above is redundant
- If there are several ways to move between two types, we consider them all
- We can do better taking an identification function followed by subtyping and supertyping (in this order)
- With some care, this leads to an algorithm that is cubic in the entity types and quadratic in the identification functions, provided that there is an upper bound on the size of type hierarchies

Faster, Faster, Faster,...

- The algorithm above is redundant
- If there are several ways to move between two types, we consider them all
- We can do better taking an identification function followed by subtyping and supertyping (in this order)
- With some care, this leads to an algorithm that is cubic in the entity types and quadratic in the identification functions, provided that there is an upper bound on the size of type hierarchies
- Essentially, it's cubic (works for thousands of entity types)

Highlights

- The type semantics above allows the same element x to belong to different type hierarchies, with a clearly defined meaning

Highlights

- The type semantics above allows the same element x to belong to different type hierarchies, with a clearly defined meaning
- This is in line with common practise in SQL databases (rows in different tables use the same numerical identifiers)

Highlights

- The type semantics above allows the same element x to belong to different type hierarchies, with a clearly defined meaning
- This is in line with common practise in SQL databases (rows in different tables use the same numerical identifiers)
- By making slightly stronger the stability condition, one can force all elements representing different entities to be distinct ("theoretician's semantics")

Highlights

- The type semantics above allows the same element x to belong to different type hierarchies, with a clearly defined meaning
- This is in line with common practise in SQL databases (rows in different tables use the same numerical identifiers)
- By making slightly stronger the stability condition, one can force all elements representing different entities to be distinct ("theoretician's semantics")
- The bicategorical composition of multirelation (pullback) is simply the join operation from relational algebra