Logic for Ontologists

John Beverley, *PhD*

Assistant Professor, *University at Buffalo*
Co-Director, *National Center for Ontological Research*
Affiliate Faculty, *Institute of Artificial Intelligence and Data Science*

Outline

- A Brief History of Logics in Ontology Engineering
- Description Logic: ALC and Extensions
- The Bisimulation Theorem

Outline

- A Brief History of Logics in Ontology Engineering
- Description Logic: ALC and Extensions
- The Bisimulation Theorem

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives
  - Predicates
  - Variables
  - Punctuation

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
    - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
    - Predicates
    - Variables
    - Punctuation

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates
  - Variables
  - Punctuation

"not"

"if…then"

"and"

"or"

"for all _"

"for some_"

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables
  - Punctuation

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables
  - Punctuation

"_ is P"
"_ is R"

"_ is P to _"
"_ is R to _"

"_ is P to _ and _"
"_ is R to _ and _"

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_) variables fill in these slots
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation

variables fill in slots for predicates too

First-Order Logic

- Researchers across the sciences typically use a formal language called *First-Order Logic* (FOL)

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation ([,], (,), {, })

Example

- English sentence: “All bald men are happy.”

Example

- English sentence: “All bald men are happy.”

- First Paraphrase: Everything that is bald and a man is happy.

Example

- English sentence: “All bald men are happy.”
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy

Example

- English sentence: "All bald men are happy."
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy
- ∀_((B_ & M_) → H_)

Example

- English sentence: “All bald men are happy.”
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy
- $\forall x((Bx \,\&\, M\_) \rightarrow H\_)$

Example

- English sentence: “All bald men are happy.”
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy

The variable associated with “V” binds variables associated with predicates within its *scope*

- ∀x((Bx & M_) → H_)

Example

- English sentence: “All bald men are happy.”
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy
- ∀x((Bx & Mx) → H_)

Example

- English sentence: "All bald men are happy."
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy
- $\forall x((Bx \mathbin{\&} Mx) \rightarrow H\_)$

Example

- English sentence: “All bald men are happy.”

- First Paraphrase: Everything that is bald and a man is happy

- Second Paraphrase: For every x, if x is bald and a man, then x is happy

In this case all the variables are within the scope of “V”

- $\forall x((Bx \,\&\, Mx) \rightarrow Hx)$

Example

- English sentence: “All bald men are happy.”
- First Paraphrase: Everything that is bald and a man is happy
- Second Paraphrase: For every x, if x is bald and a man, then x is happy
- $\forall x((Bx \mathbin{\&} Mx) \rightarrow Hx)$

Interpretation

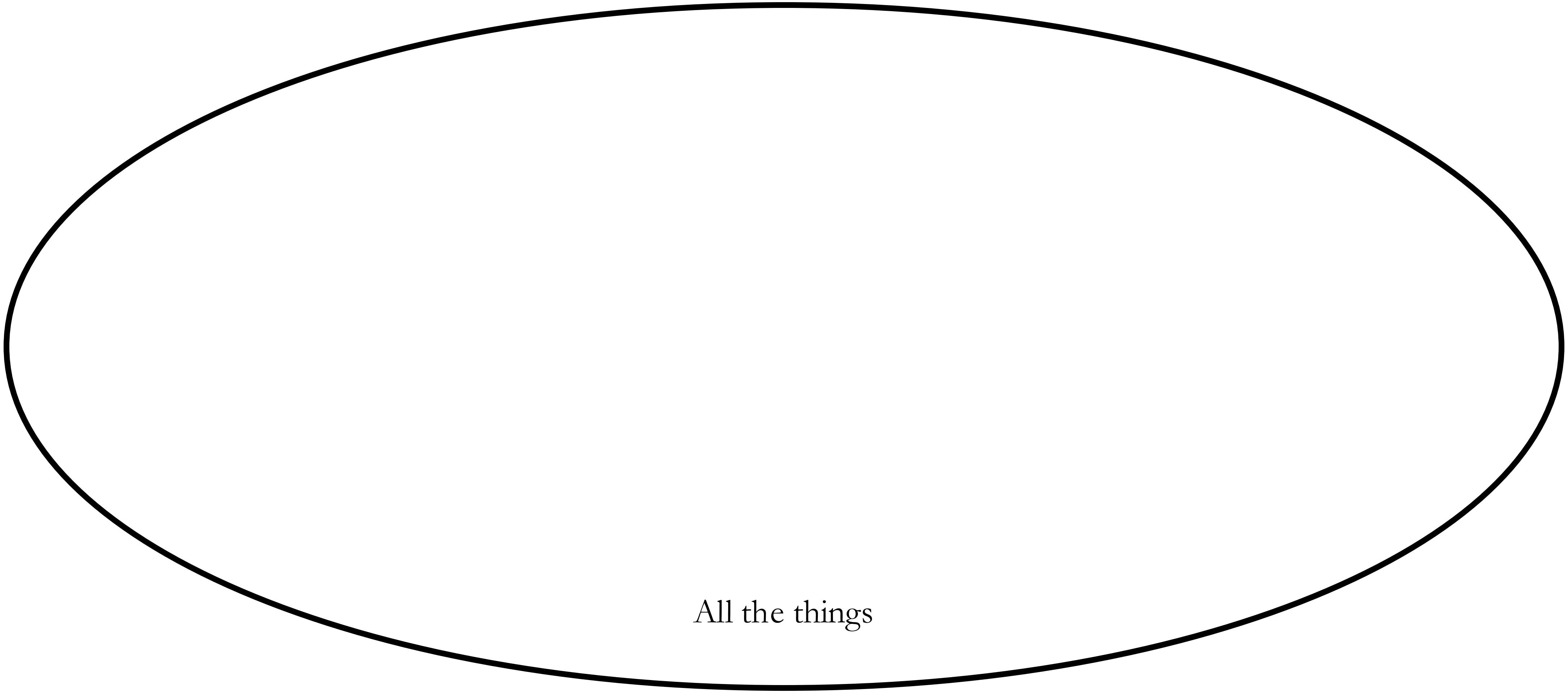
For every x, if x is bald and a man, then x is happy

Interpretation

For every x, if x is bald and a man, then x is happy

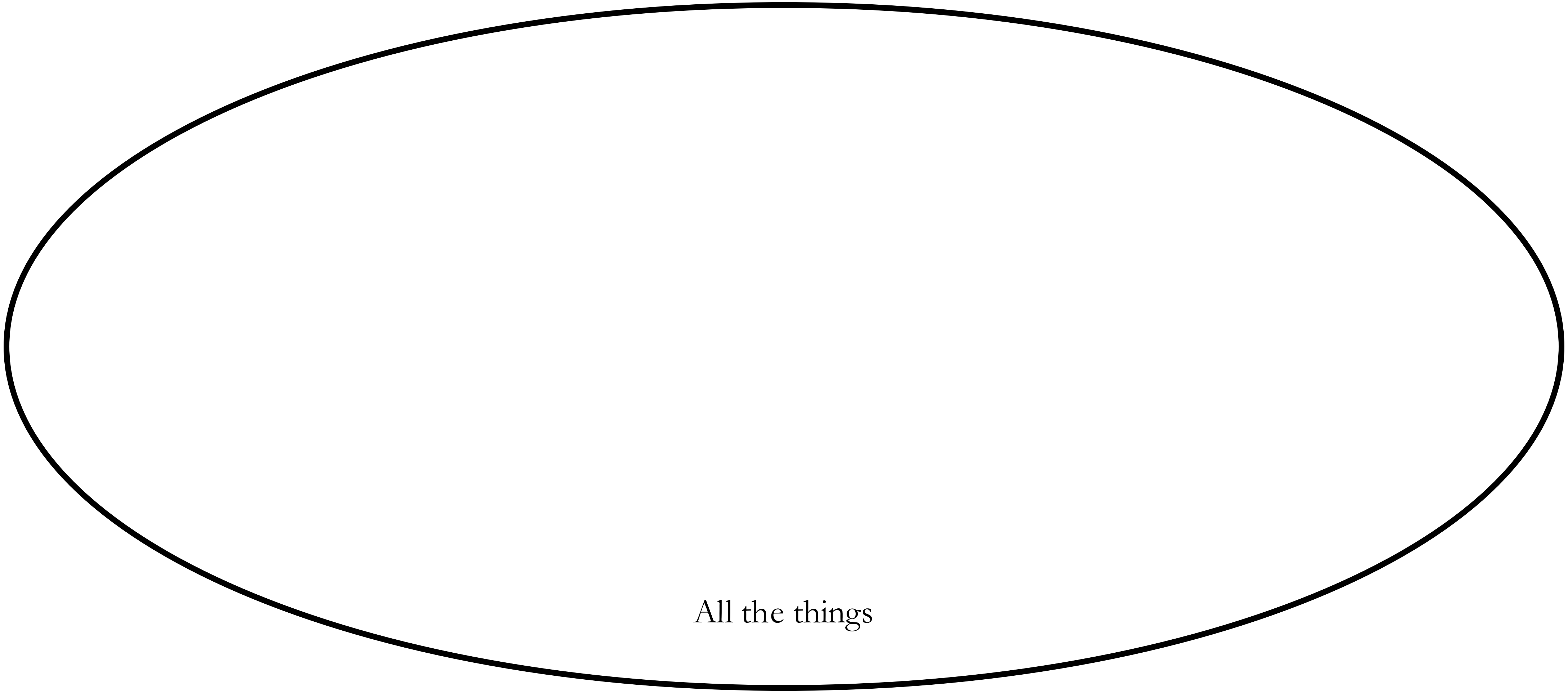
Interpretation

For every x, if x is bald and a man, then x is happy



Interpretation

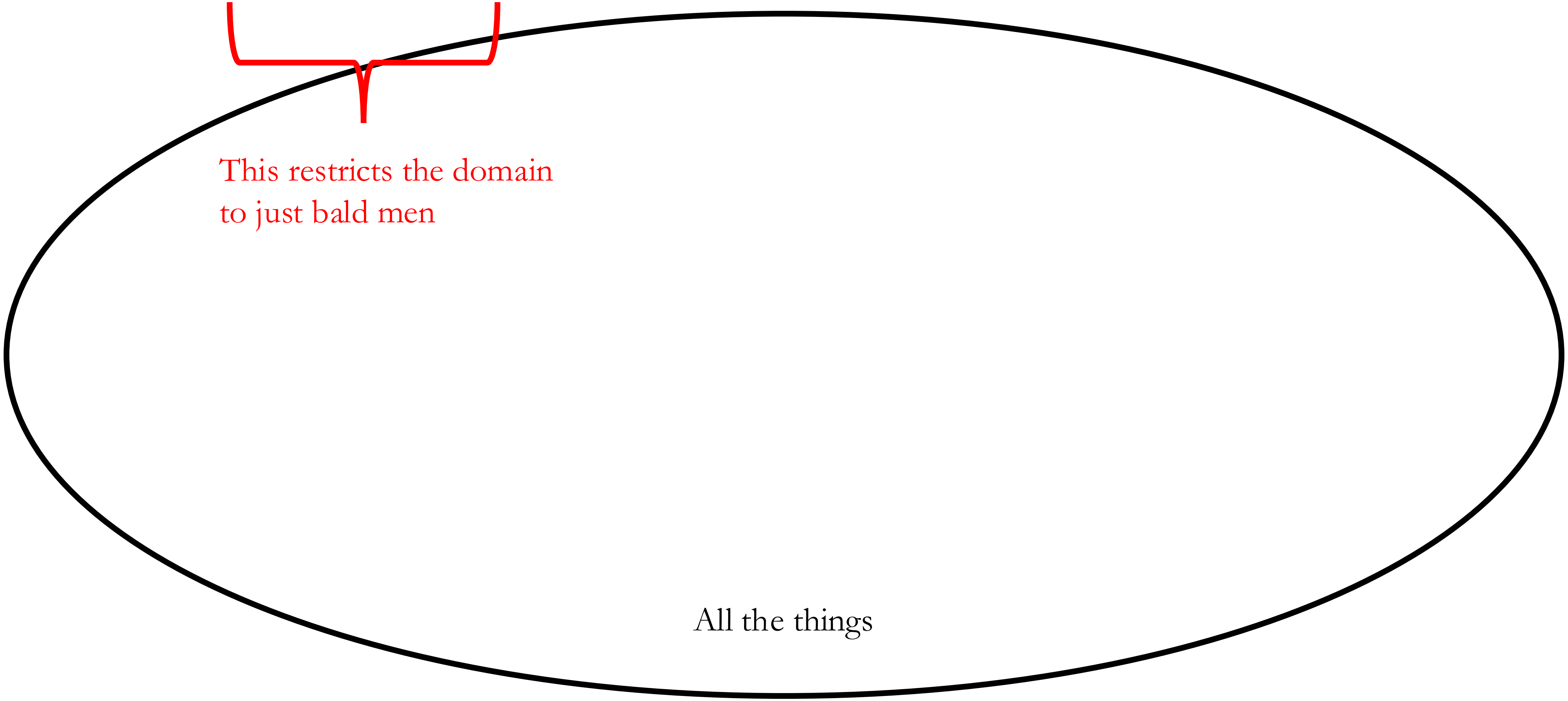
For every x, if x is bald and a man, then x is happy



Interpretation

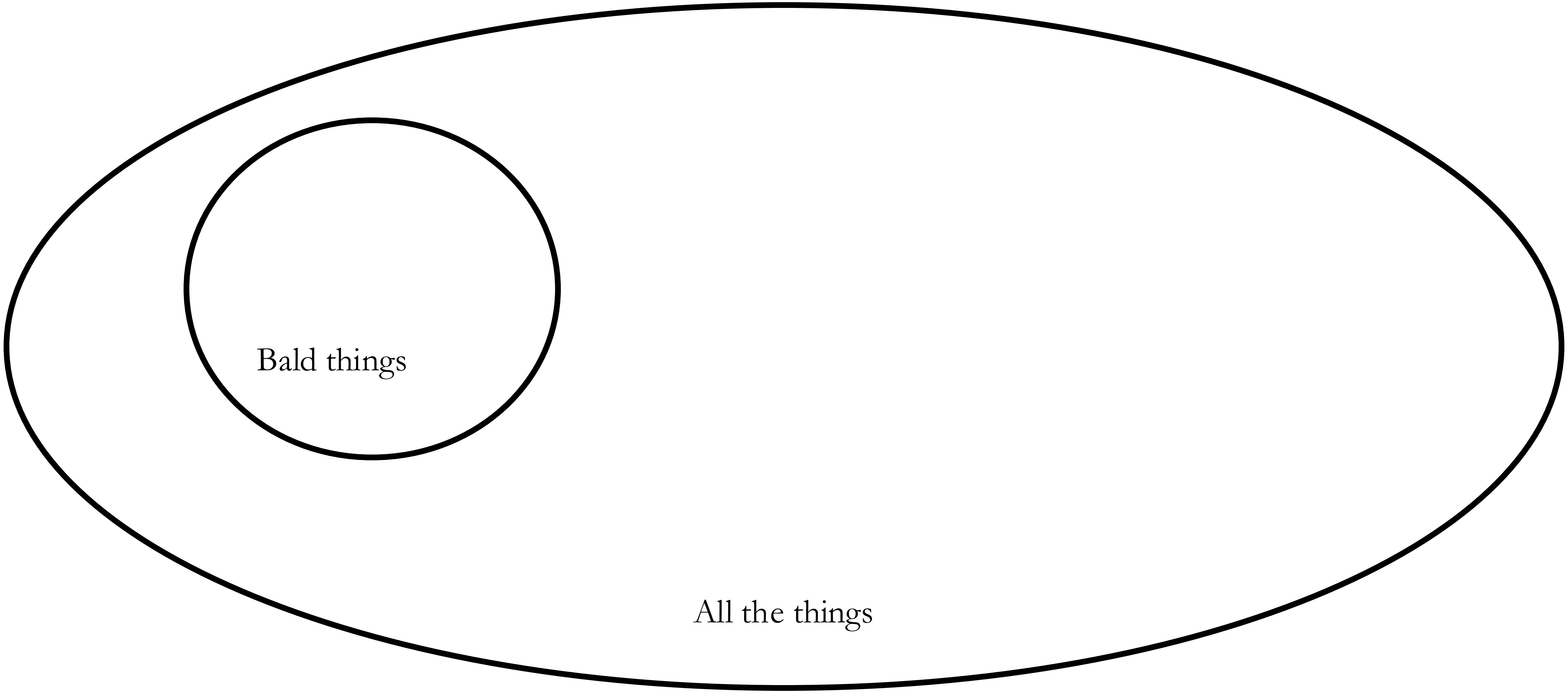
For every x, if x is bald and a man, then x is happy

This restricts the domain
to just bald men



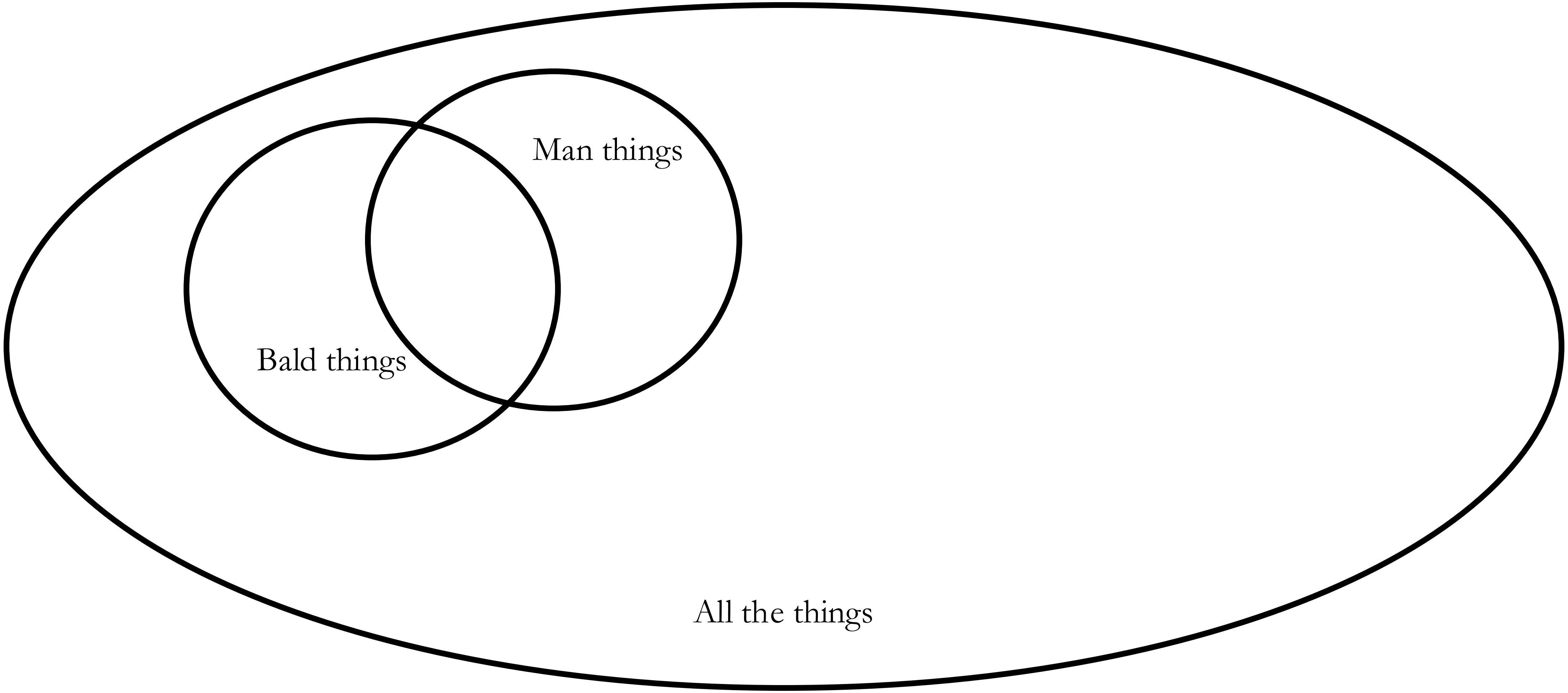
Interpretation

For every x, if x is bald and a man, then x is happy



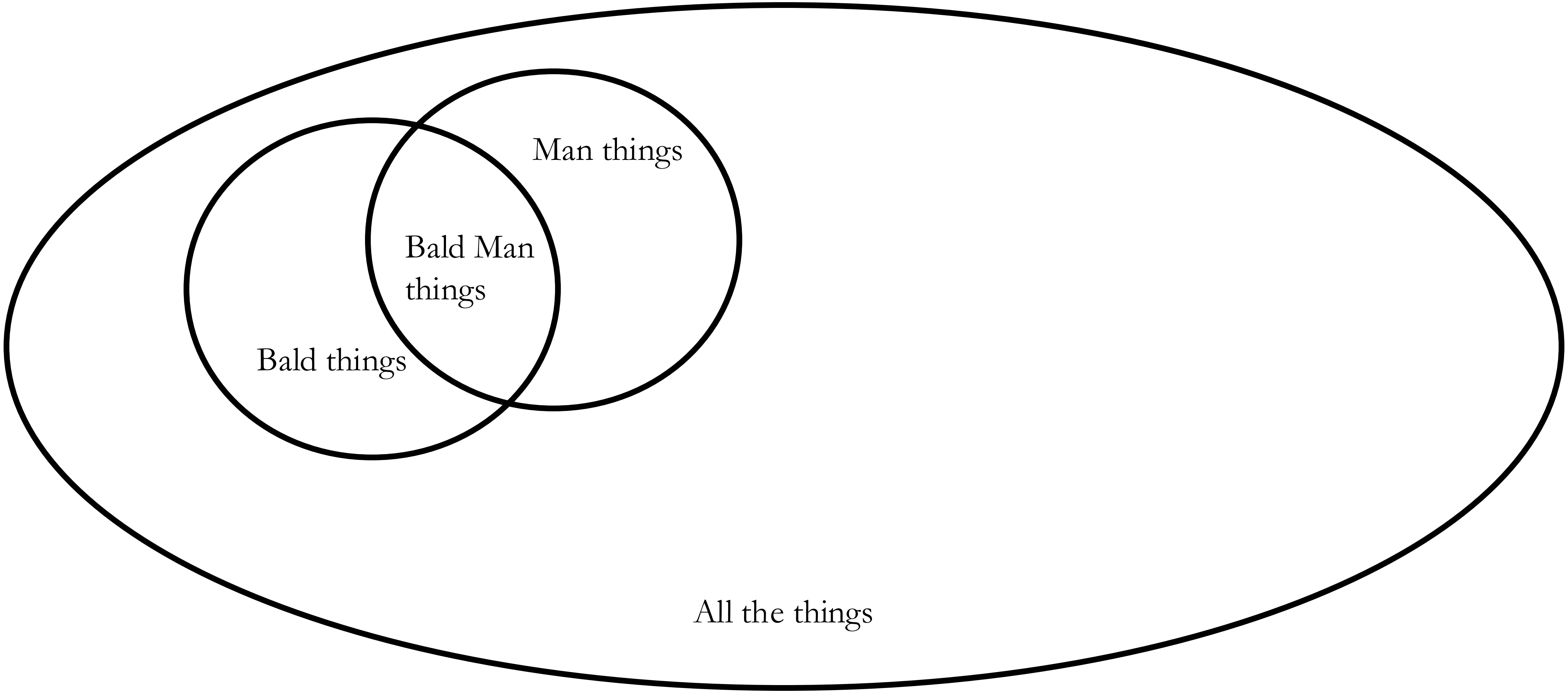
Interpretation

For every x, if x is bald and a man, then x is happy



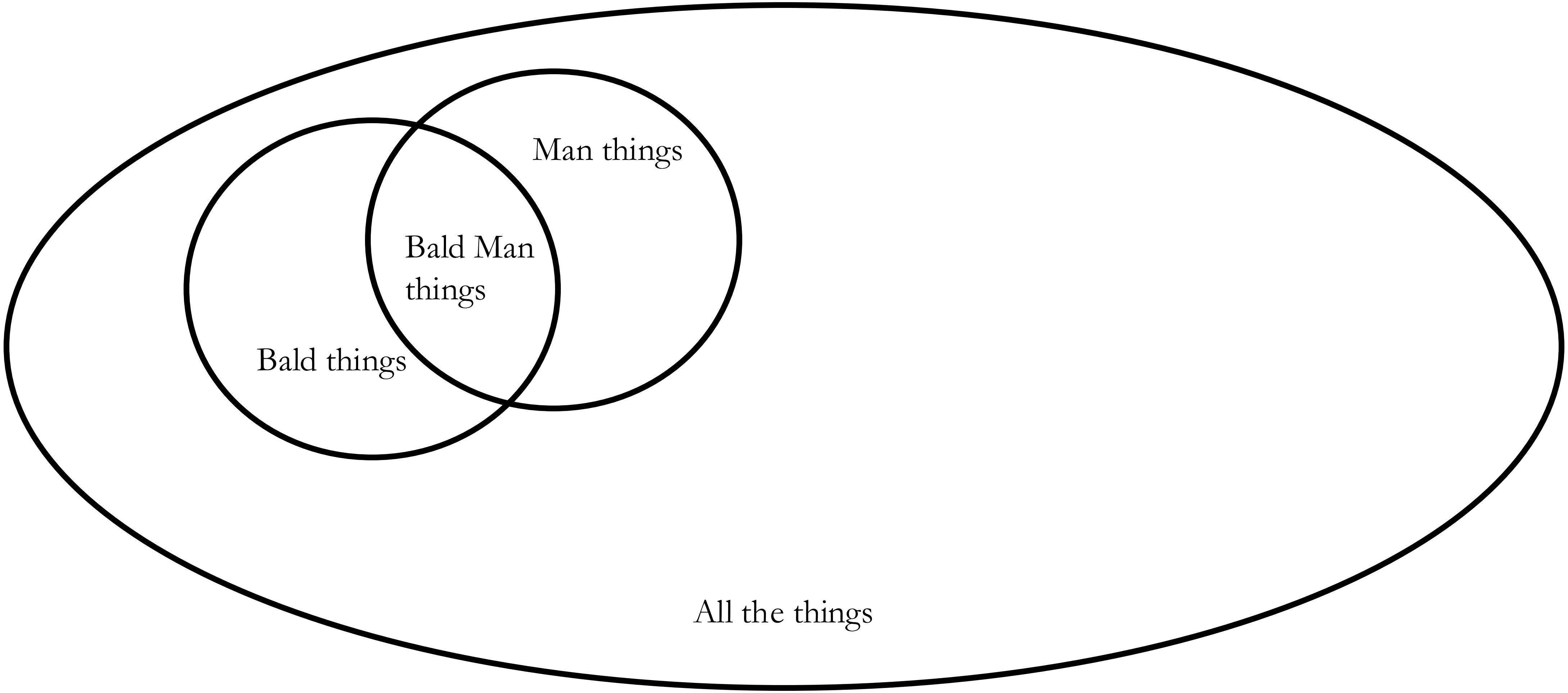
Interpretation

For every x, if x is bald and a man, then x is happy



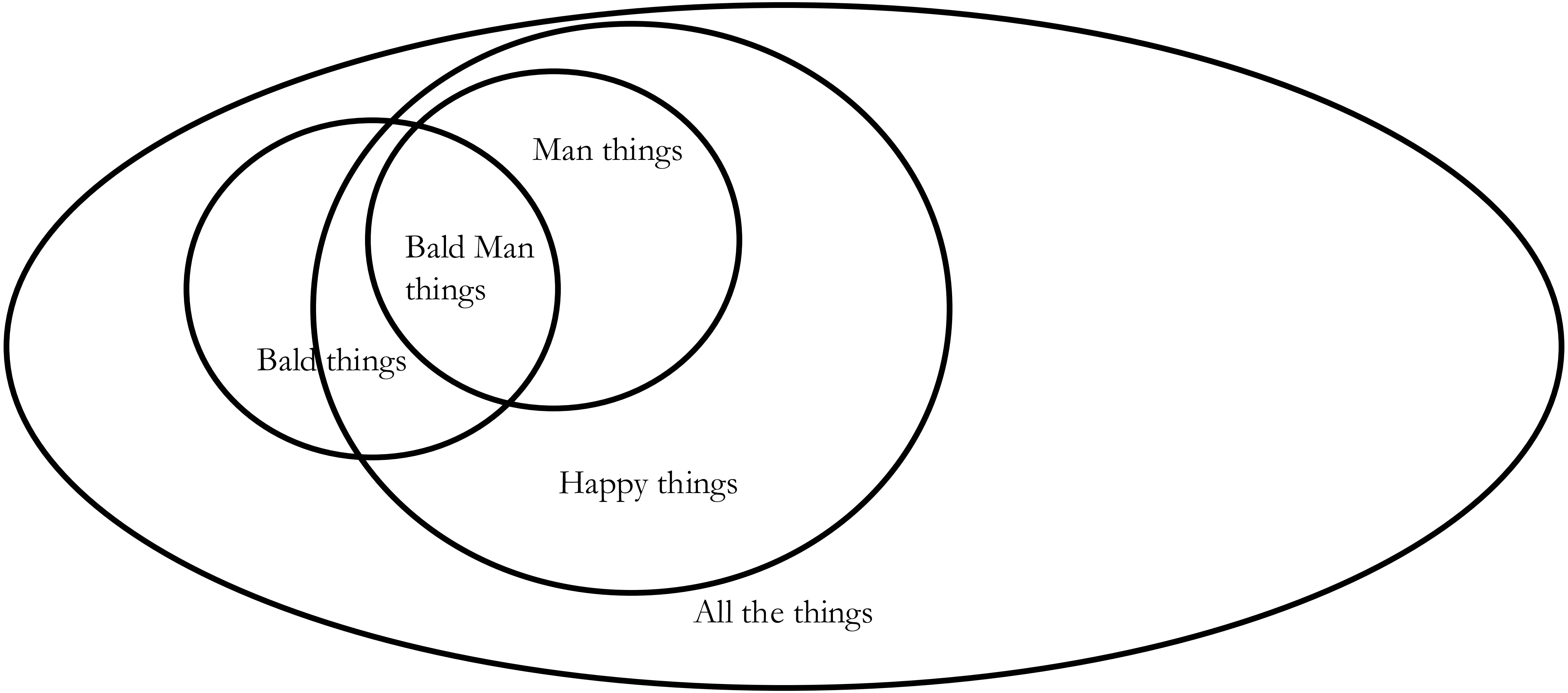
Interpretation

For every x, if x is bald and a man, then x is happy



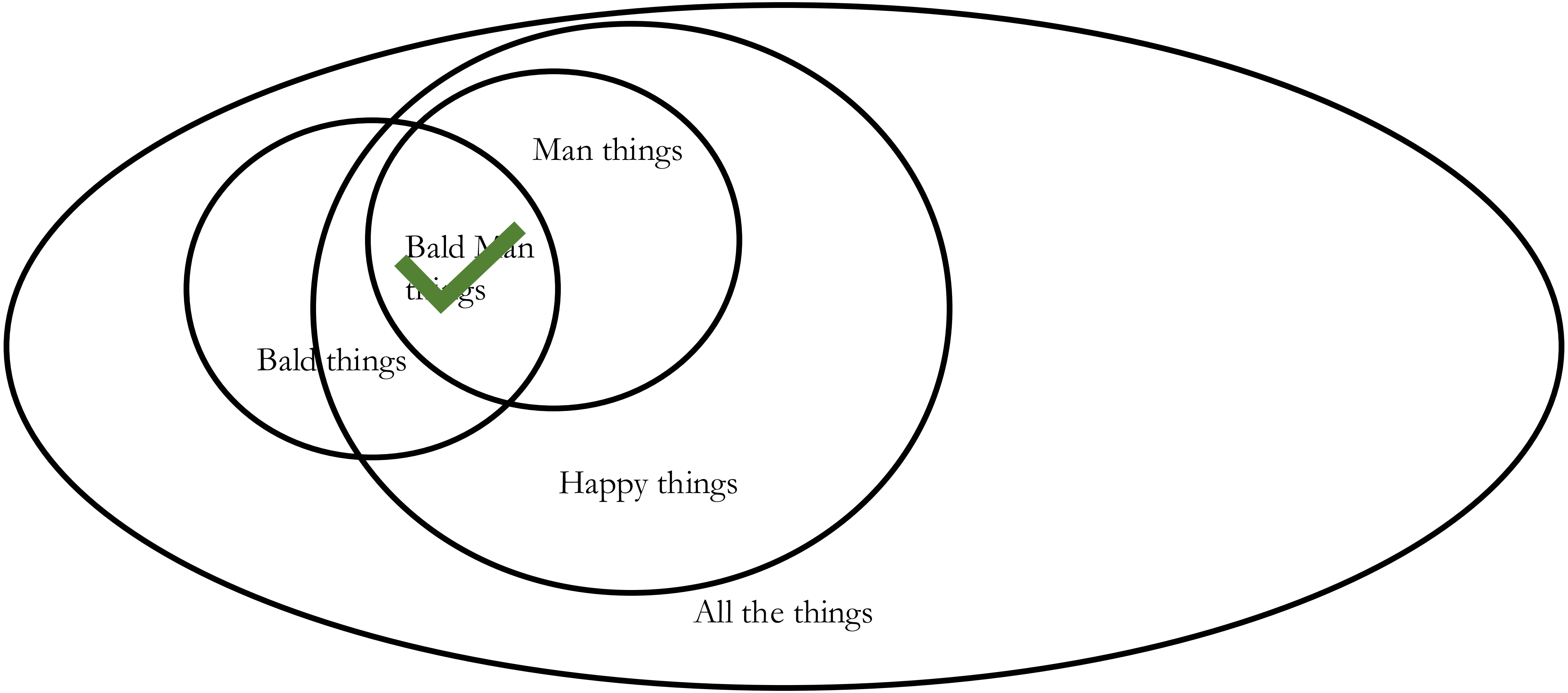
Interpretation

For every x, if x is bald and a man, then x is happy



Interpretation

For every x, if x is bald and a man, then x is happy



Supplemented FOL

- Researchers frequently add *names* to the FOL language

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation ([,], (,), {, })

Supplemented FOL

- Researchers frequently add *names* to the FOL language

- FOL consists of the following vocabulary:
    - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
    - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
    - Variables (x, y, z, …)
    - Punctuation ([,], (,), {, })
    - Names

Supplemented FOL

- Researchers frequently add *names* to the FOL language

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation ([,], (,), {, })
  - Names (a, b, c, d, …)

Supplemented FOL

- Researchers frequently add *names* to the FOL language

- FOL consists of the following vocabulary:
  - Logical operators and connectives (~, →, &, v, ∀_, ∃_)
  - Predicates (P_, R_, …P_ _, R_ _,…P_ _ _, R_ _ _, …)
  - Variables (x, y, z, …)
  - Punctuation ([,], (,), {, })
  - Names (a, b, c, d, …)

"Alex"
"Bob"
"Chris"
"Deborah"

Supplemented FOL

- Compare:

- Someone is bald and happy
  - **Ex(Bx & Hx)**

- John is bald and happy
  - **(Bj & Hj)**

Binary Relations

- FOL includes predicates, e.g. *is red*, *is bald*, and relations, e.g. *is part of*, *is between*, *is next to*, etc.

- For example (give John's arm the name 'a'):
  - John has an arm and it is part of John
  - part of(a, j)
  - John has a sister, Kellye
  - is related to(j, k)
  - John is between Sam and Deborah
  - is between (j, s, d)

Pervasiveness

- FOL is *very* expressive, which is one of the reasons that it is used so widely as the formal language underpinning science

- You may have heard that the language of *mathematics* underwrites all of modern science...

- Note, **contemporary mathematics is written in First-Order Logic**

Theorem:

$(\forall x \mid: P \Rightarrow Q) \quad \equiv \quad \{x \mid P\} \subseteq \{x \mid Q\}$

Proof:

$$
\begin{aligned}
& \{x \mid P\} \subseteq \{x \mid Q\} \\
= \quad & \langle \text{Def. of Subset } \subseteq \text{, with } v \text{ not occurring free in } P \text{ or } Q \rangle \\
& (\forall v \mid v \in \{x \mid P\} : v \in \{x \mid Q\}) \\
= \quad & \langle v \in \{x \mid R\} \;\equiv\; R[x := v] \text{, twice} \rangle \\
& (\forall v \mid P[x := v] : Q[x := v]) \\
= \quad & \langle \text{Trading; Dummy renaming} \rangle \\
& (\forall x \mid: P[x := v][v := x] \Rightarrow \\
& \qquad Q[x := v][v := x]) \\
= \quad & \langle R[x := v][v := x] \;\equiv\; R \text{ if } v \text{ does not occur free in } R \text{, twice} \rangle \\
& (\forall x \mid: P \Rightarrow Q)
\end{aligned}
$$

FOL is Too Expressive…

- Once we add **ternary relations**, the formal language becomes **undecidable**

- What this means – roughly – is that we can't determine in FOL for every expression whether that expression is **false** or we **just haven't found a counterexample to it**

- Put another way, it's impossible in FOL to construct a step-by-step procedure whose evaluation for truth or falsity **always** terminates for **any** FOL sentence

Restricting FOL

- Undecidability arises with respect to FOL because it is too expressive
- This observation led computer scientists to reflect on how the expressivity of FOL might be restricted so that the resulting language was **decidable**
- The idea then was to find the **most expressive version** of FOL that would not result in undecidability

Restricting FOL

- This research program – started in the 1980s – continues to this day, with varying degrees of success
- Notably, it spurred the creation of new logics based on FOL, called **description logics**
- Description logics are a **family of logics designed to be expressive decidable languages based on FOL**

Web Ontology Language

- Most relevant to us ontology engineers is that description logics provide the theoretical foundation on which the **Web Ontology Language (OWL)** is based

- In the remainder of this lesson, we will explore description logics of varying degrees of expressivity

- Thereby providing theoretical understanding of OWL, which will occupy much of our subsequent discussion

Outline

- A Brief History of Logics in Ontology Engineering
- Description Logic: ALC and Extensions
- The Bisimulation Theorem

Description Logics

- Basically, description logics start with FOL, and add the following:
- Relations of no greater than arity of 2 are allowed
- The same object may have multiple names
- Not asserting something does not mean it is false

Description Logics

- Basically, description logics start with FOL, and add the following:

- Relations of no greater than arity of 2 are allowed

Binary Fragment of FOL

- The same object may have multiple names

- Not asserting something does not mean it is false

Description Logics

- Basically, description logics start with FOL, and add the following:

- Relations of no greater than arity of 2 are allowed

Binary Fragment of FOL

- The same object may have multiple names

Unique Name Assumption

- Not asserting something does not mean it is false

Description Logics

- Basically, description logics start with FOL, and add the following:

- Relations of no greater than arity of 2 are allowed

  Binary Fragment of FOL

- The same object may have multiple names

  Unique Name Assumption

- Not asserting something does not mean it is false

  Open World Assumption

ALC Syntax

Definition 2.1. Let $\mathbf{C}$ be a set of *concept names* and $\mathbf{R}$ be a set of *role names* disjoint from $\mathbf{C}$. The set of $\mathcal{ALC}$ *concept descriptions* over $\mathbf{C}$ and $\mathbf{R}$ is inductively defined as follows:

- Every concept name is an $\mathcal{ALC}$ concept description.
- $\top$ and $\bot$ are $\mathcal{ALC}$ concept descriptions.
- If C and D are $\mathcal{ALC}$ concept descriptions and r is a role name, then the following are also $\mathcal{ALC}$ concept descriptions:

  $C \sqcap D$ (conjunction),
  $C \sqcup D$ (disjunction),
  $\neg C$ (negation),
  $\exists r.C$, (existential restriction), and
  $\forall r.C$ (value restriction).

ALC

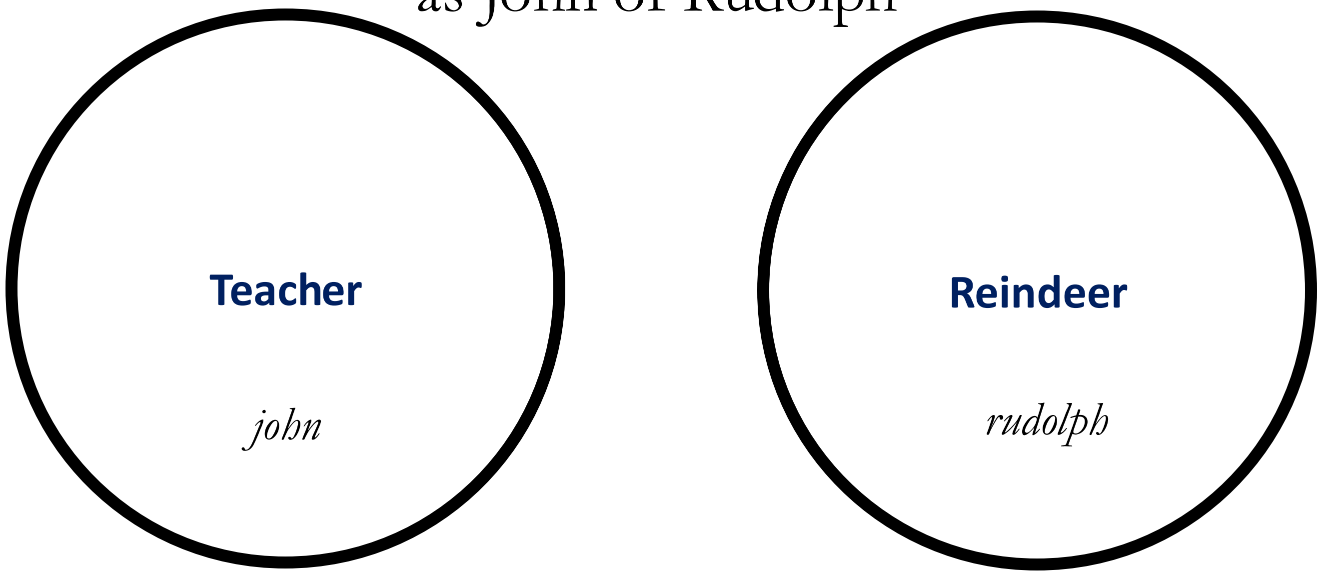
Signature = $\{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$

- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - $\top$
  - $\bot$
  - $C \sqcup D$
  - $C \sqcap D$
  - $\neg C$
  - $\exists r.C$
  - $\forall r.C$

ALC

Signature = $\{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$

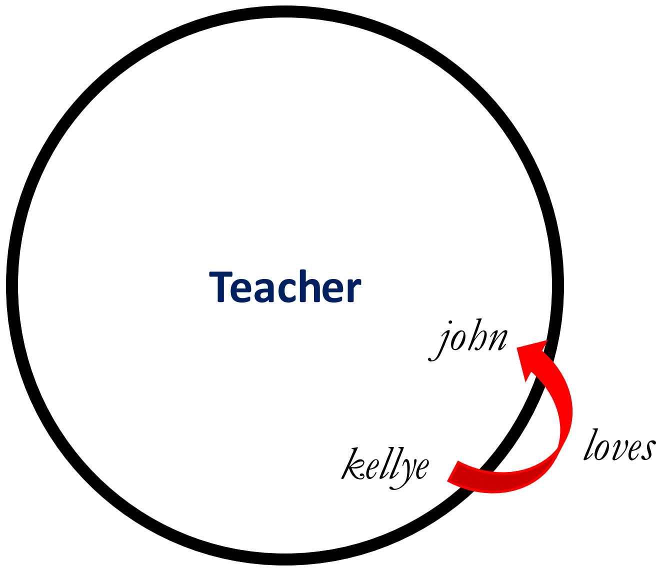
- Concept Descriptions:
  - $C_{1\ldots n}$ - Correspond to classes, such as Teacher, Reindeer, etc. which are often assumed to be collections of similar enough instances in the world, such as John or Rudolph



ALC

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

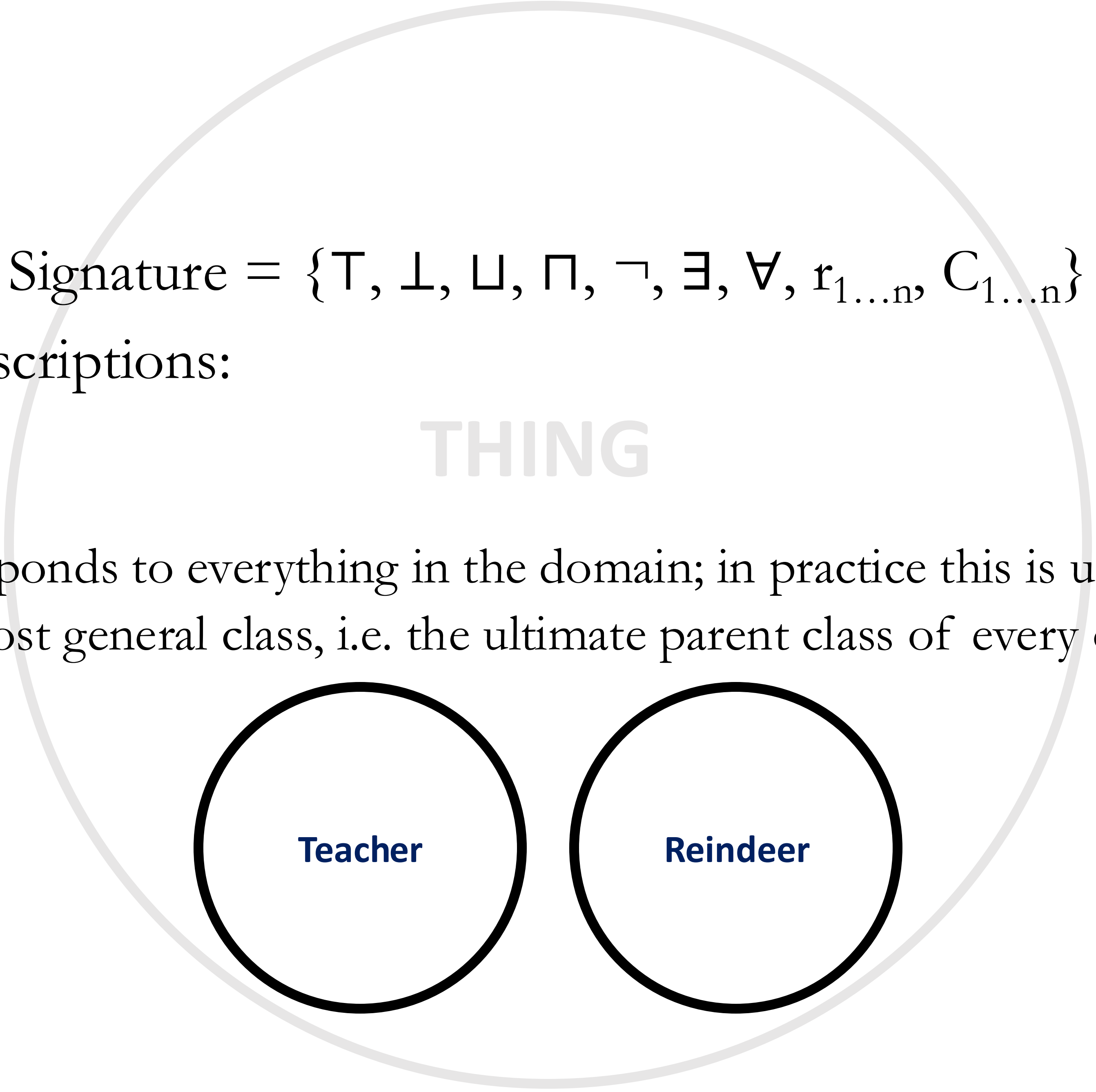
- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$ - Corresponds to relations holding between instances such as loves or parent of or next to



ALC

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - $\top$ - Corresponds to everything in the domain; in practice this is used to represent the most general class, i.e. the ultimate parent class of every other class

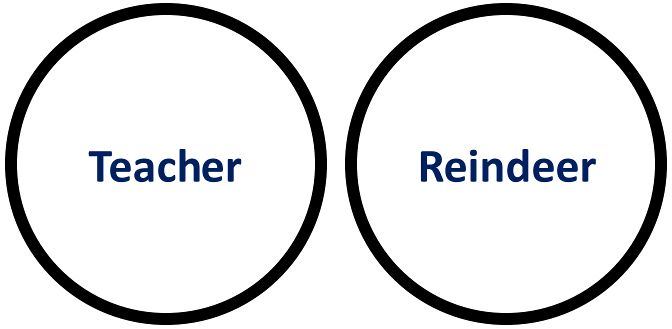


ALC

Signature = {⊤, ⊥, ⊔, ⊓, ¬, ∃, ∀, $r_{1\ldots n}$, $C_{1\ldots n}$}

- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - ⊤
  - ⊥ - Corresponds to nothing in the domain; in practice this is used to represent the least general class, i.e. the class that contains nothing

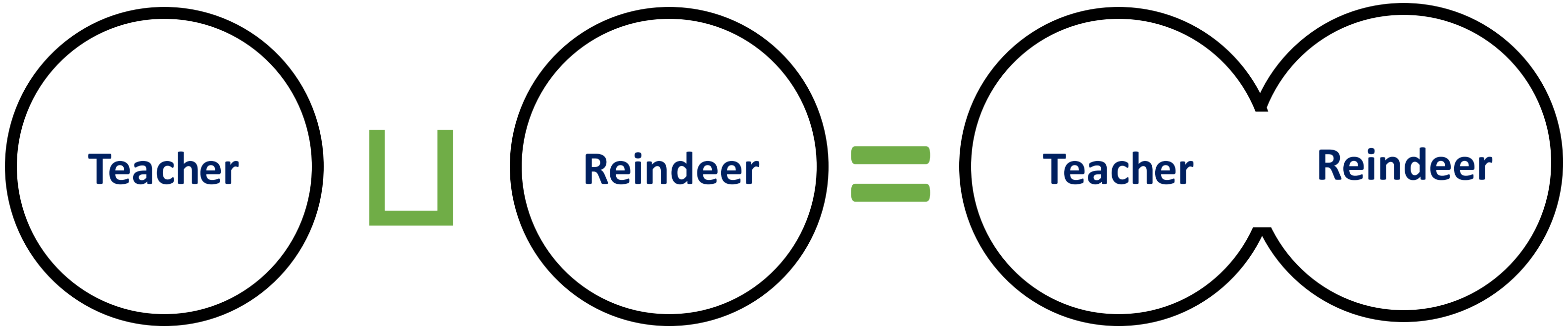
NOTHING



ALC

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

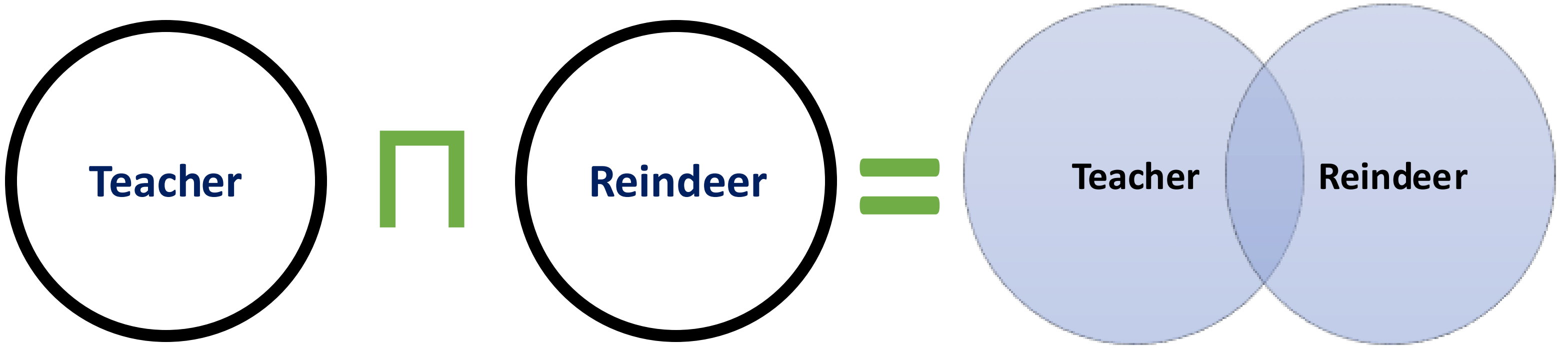
- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - $\top$
  - $\bot$
  - $C \sqcup D$ – Corresponds to the grouping of all instances of class C with all the instances of class D; this is sometimes called (imprecisely) the *union* of C and D



ALC

Signature = $\{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$

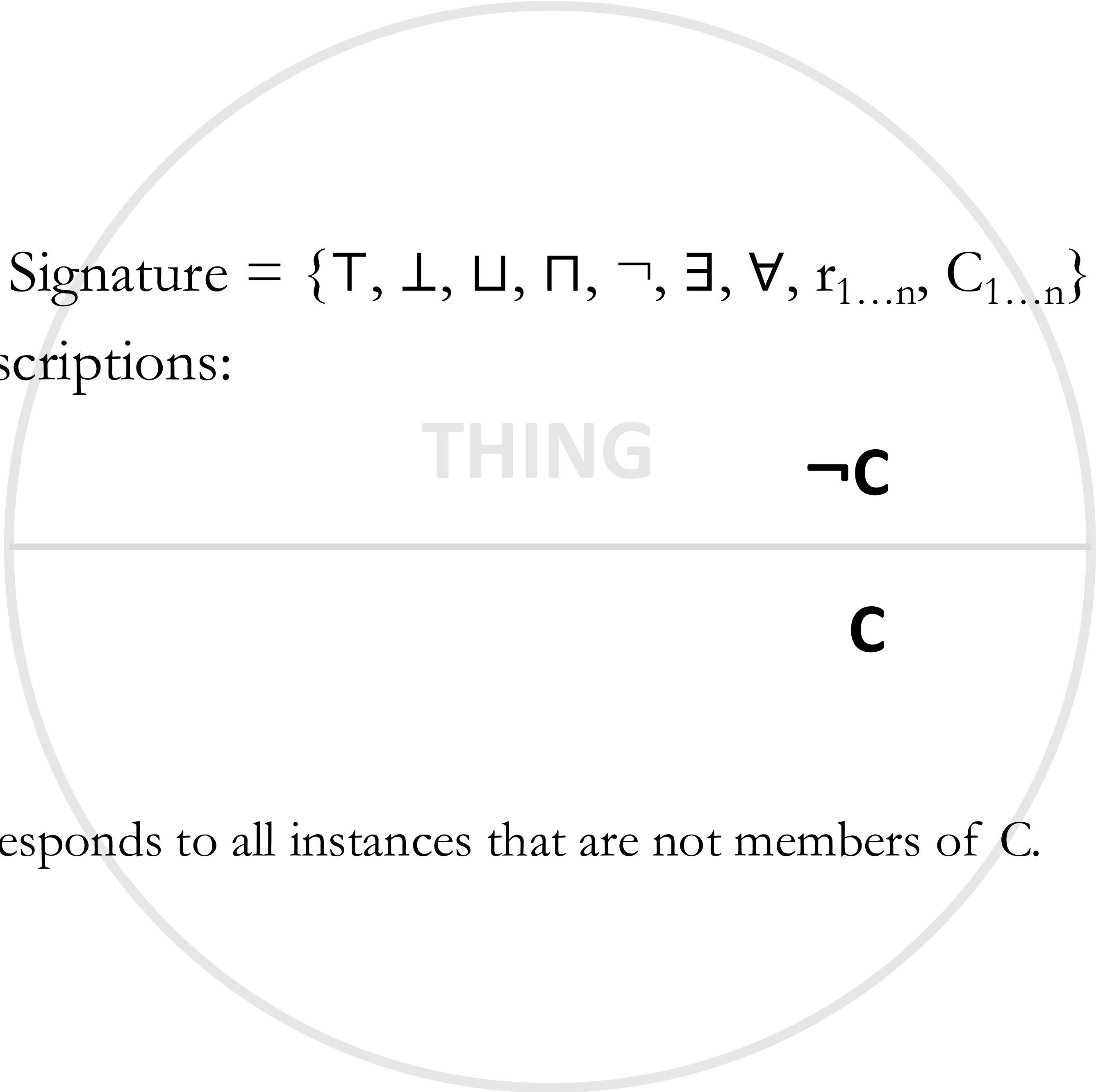
- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - $\top$
  - $\bot$
  - $C \sqcup D$
  - $C \sqcap D$ – Corresponds to all instances that are members of both C and D; sometimes called (imprecisely) the *intersection* of C and D



ALC

Signature = {⊤, ⊥, ⊔, ⊓, ¬, ∃, ∀, $r_{1\ldots n}$, $C_{1\ldots n}$}

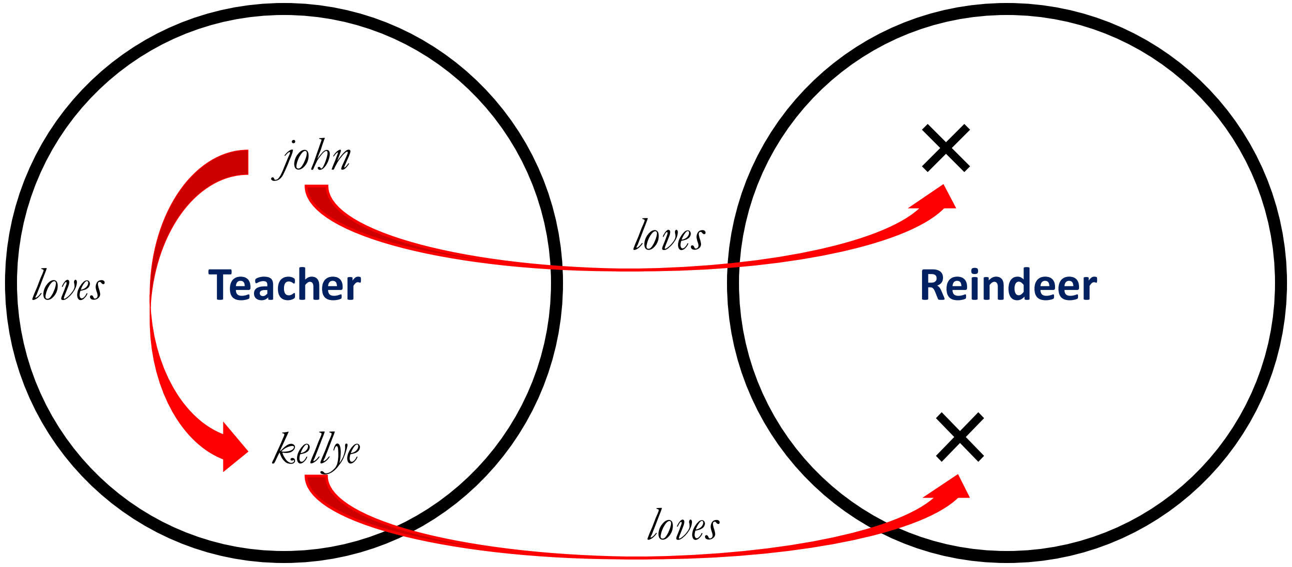
- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - ⊤
  - ⊥
  - C ⊔ D
  - C ⊓ D
  - ¬C – Corresponds to all instances that are not members of C.



ALC

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

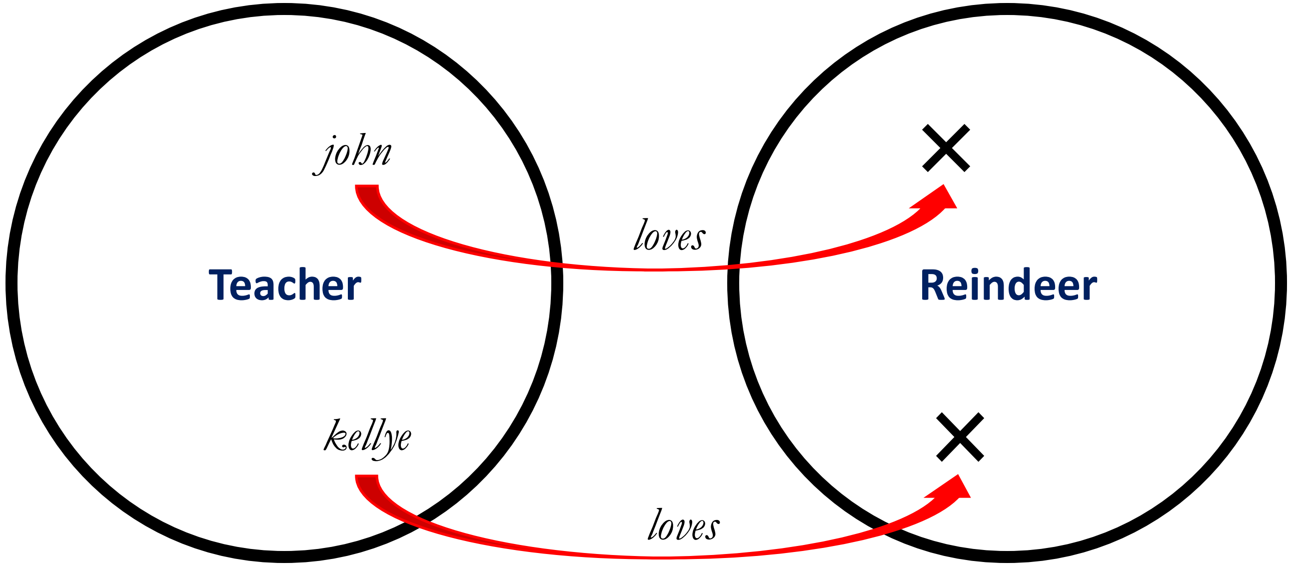
- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - $\top$
  - $\bot$
  - $C \sqcup D$
  - $C \sqcap D$
  - $\neg C$
  - $\exists r.C$ – Corresponds to all instances that are related to some C.



ALC

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

- Concept Descriptions:
  - $C_{1\ldots n}$
  - $r_{1\ldots n}$
  - $\top$
  - $\bot$
  - $C \sqcup D$
  - $C \sqcap D$
  - $\neg C$
  - $\exists r.C$
  - $\forall r.C$ – Corresponds to all instances that are related to only C.



ALC Extensions: ALCI (inverses)

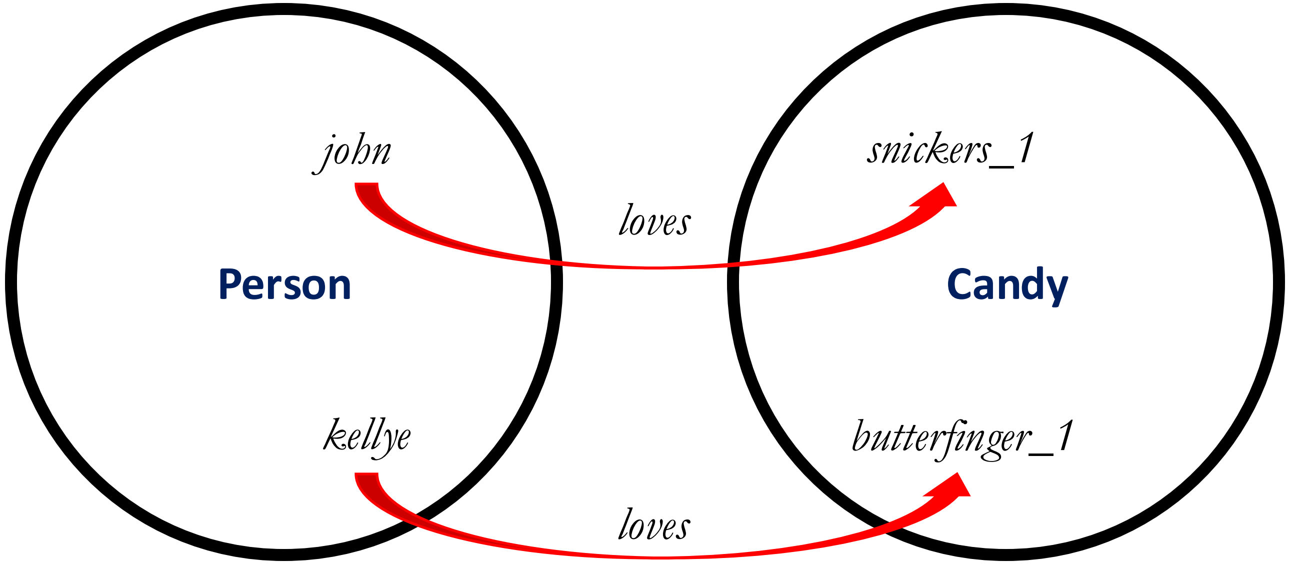
ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves⁻', i.e. 'loved by'

ALC Extensions: ALCI (inverses)

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

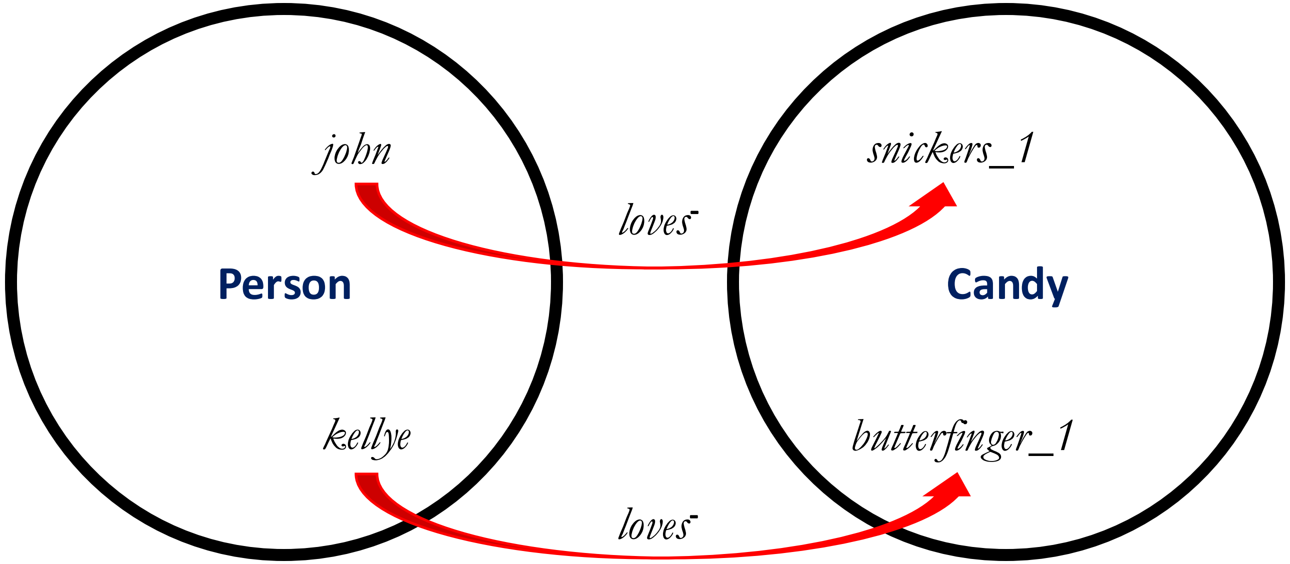
- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of ‘loves’ being ‘loves⁻’, i.e. ‘loved by’



ALC Extensions: ALCI (inverses)

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

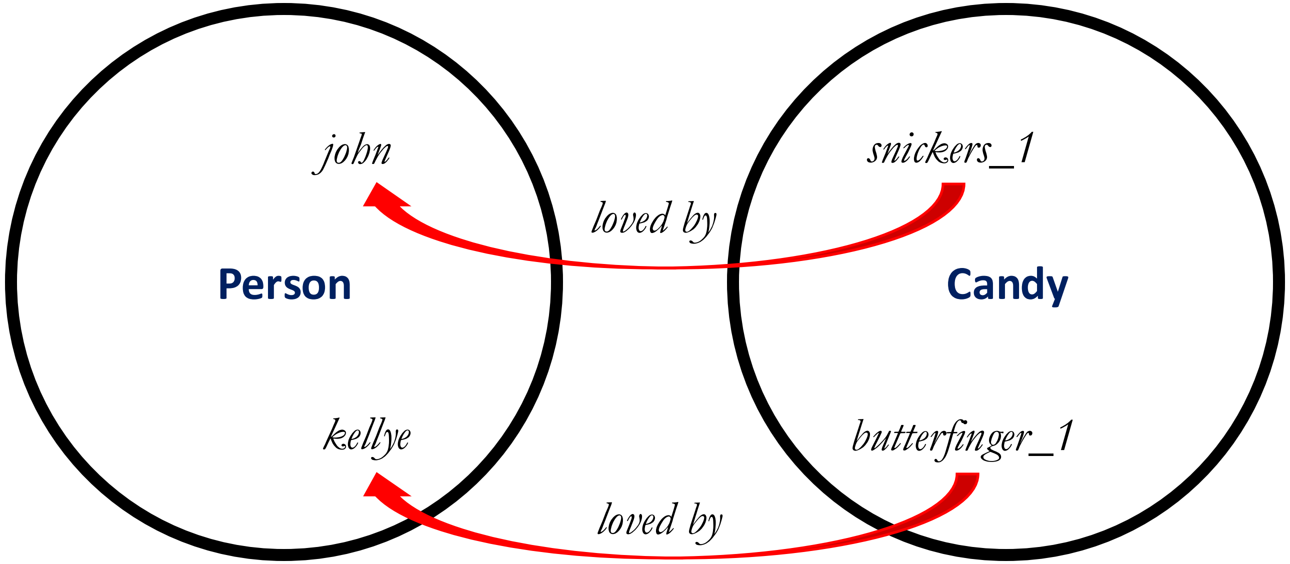
- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves⁻', i.e. 'loved by'



ALC Extensions: ALCI (inverses)

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of ‘loves’ being ‘loves⁻’, i.e. ‘loved by’



ALC Extensions: ALCN (cardinality)

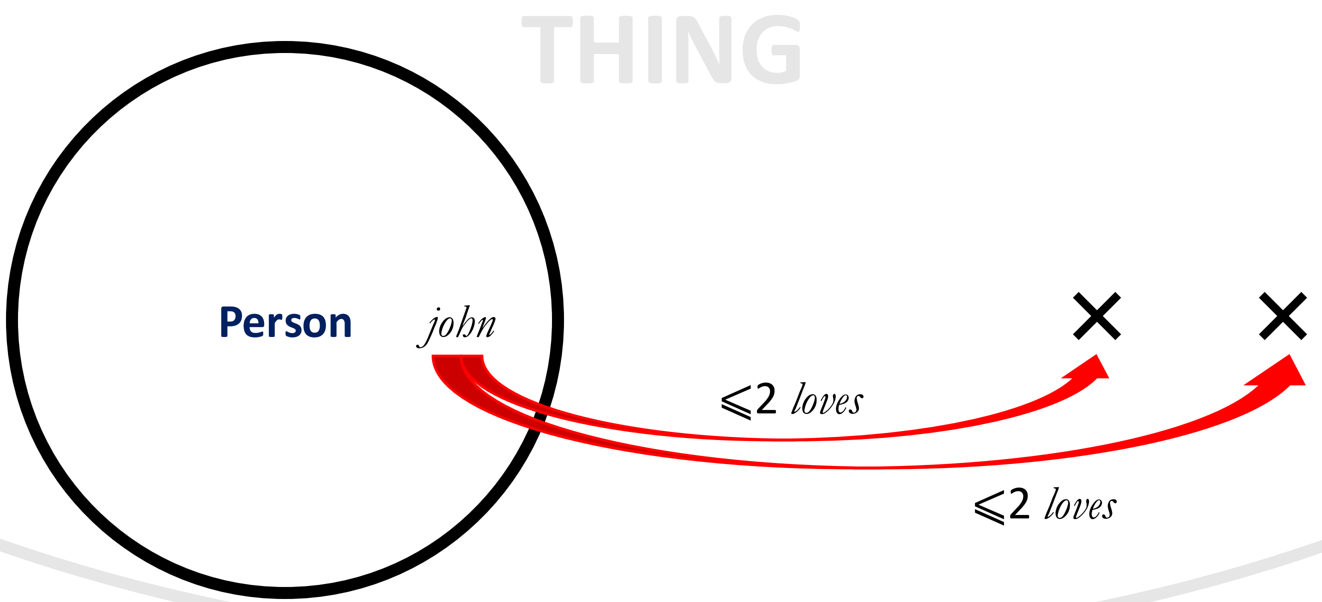
ALCN Signature = ALC Signature + {⩽n r, ⩾n r}

- ⩽n r – Corresponds to restriction that r is related to no more than n instances
- ⩾n r – Corresponds to restriction that r is related to no fewer than n instances

ALC Extensions: ALCN (cardinality)

ALCN Signature = ALC Signature + {⩽n r, ⩾n r}

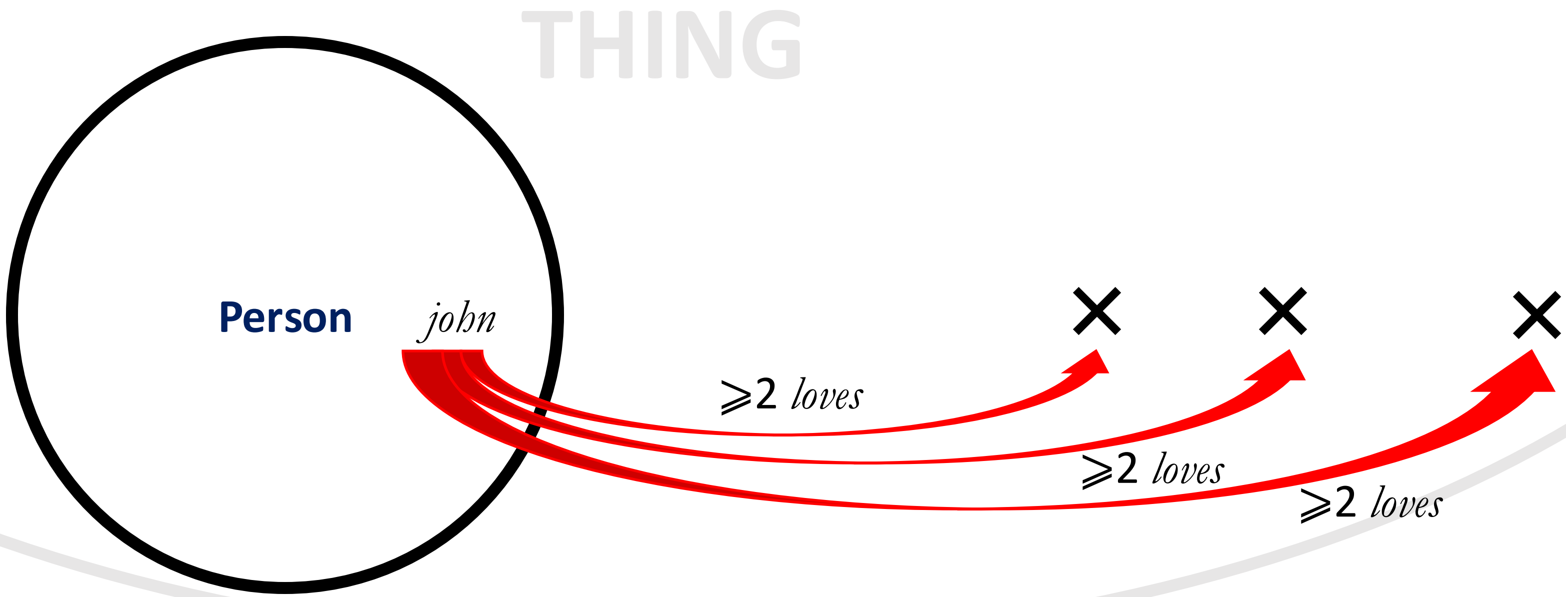
- ⩽n r – Corresponds to restriction that r is related to no more than n instances
- ⩾n r – Corresponds to restriction that r is related to no fewer than n instances



ALC Extensions: ALCN (cardinality)

ALCN Signature = ALC Signature + {⩽n r, ⩾n r}

- ⩽n r – Corresponds to restriction that r is related to no more than n instances
- ⩾n r – Corresponds to restriction that r is related to no fewer than n instances



ALC Extensions: ALCQ (qual. cardinality)

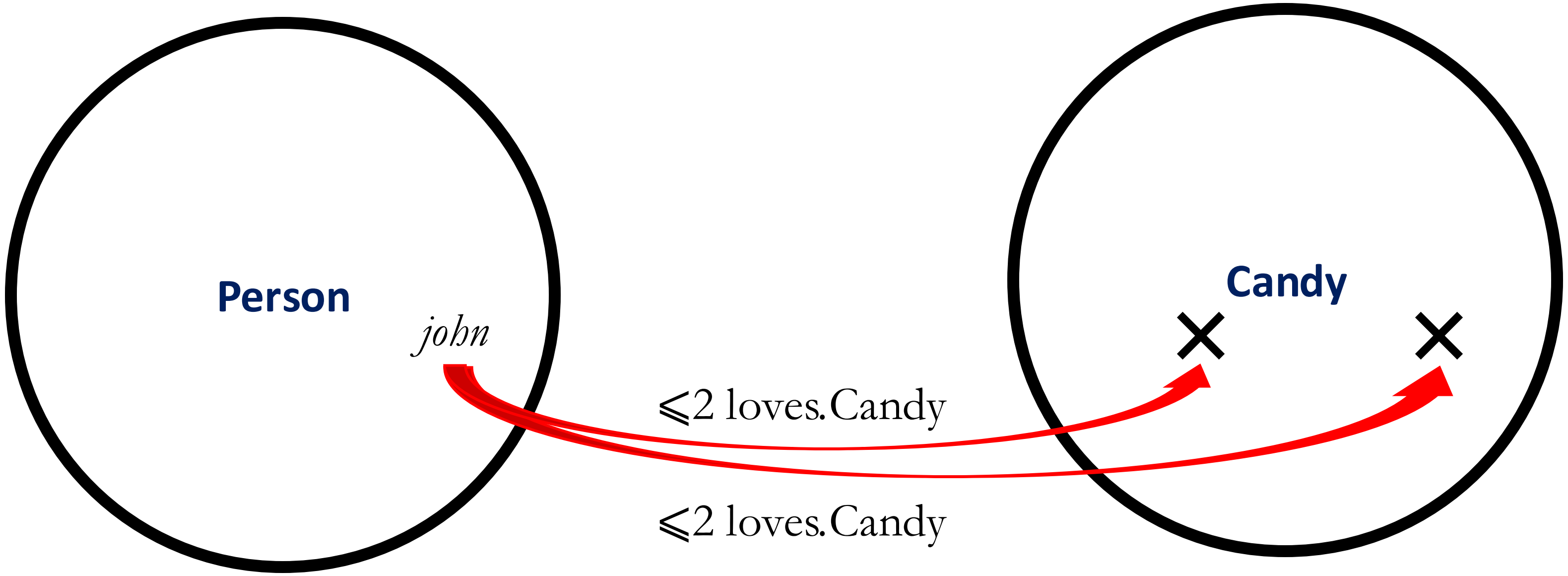
ALCQ Signature = ALC Signature + {$\leqslant$n r.C, $\geqslant$n r.C}

- $\leqslant$n r.C – Corresponds to restriction that r is related to no more than n Cs
- $\geqslant$n r.C – Corresponds to restriction that r is related to no fewer than n Cs

ALC Extensions: ALCQ (qual. cardinality)

ALCQ Signature = ALC Signature + {$\leqslant$n r.C, $\geqslant$n r.C}

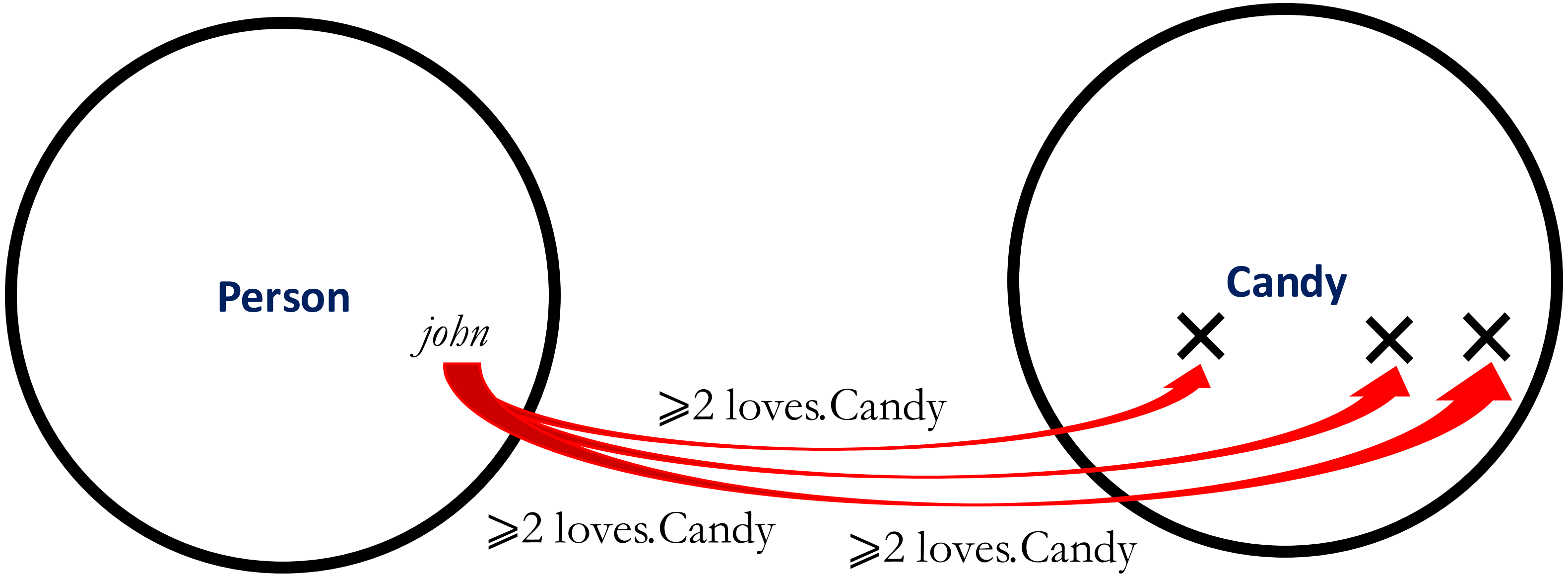
- $\leqslant$n r.C – Corresponds to restriction that r is related to no more than n Cs
- $\geqslant$n r.C – Corresponds to restriction that r is related to no fewer than n Cs



ALC Extensions: ALCQ (qual. cardinality)

ALCQ Signature = ALC Signature + {⩽n r.C, ⩾n r.C}

- ⩽n r.C – Corresponds to restriction that r is related to no more than n Cs
- ⩾n r.C – Corresponds to restriction that r is related to no fewer than n Cs



ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

  - In ALC, the connective $\sqcup$ can be used to combine *classes*

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

  - In ALC, the connective ⊔ can be used to combine *classes*

    Great Bands = The Beatles ⊔ The Eagles ⊔ Metallica ⊔…

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

  - In ALC, the connective ⊔ can be used to combine *classes*

    Great Bands = The Beatles ⊔ The Eagles ⊔ Metallica ⊔…

  - But no native connectives link *instances*

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

  - In ALC, the connective ⊔ can be used to combine *classes*

    Great Bands = The Beatles ⊔ The Eagles ⊔ Metallica ⊔…

  - But no native connectives link *instances*

    Beatles = john ? paul ? ringo ? george

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALCO, treating instances as singleton classes permits using ⊔

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALCO, treating instances as singleton classes permits using ⊔

    The Beatles = {john} ⊔ {paul} ⊔ {ringo} ⊔ {george}

ALC Extensions: ALCO (nominal)

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALCO, treating instances as singleton classes permits using ⊔

    The Beatles = {john} ⊔ {paul} ⊔ {ringo} ⊔ {george}
  - Since ⊔ can only be used to combine classes

ALC Semantics

Definition 2.2. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *interpretation domain*, and a mapping $\cdot^{\mathcal{I}}$ that maps

- every concept name $A \in \mathbf{C}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and
- every role name $r \in \mathbf{R}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{there is an } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}, \\
(\forall r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}},\ \text{if } (d,e) \in r^{\mathcal{I}},\ \text{then } e \in C^{\mathcal{I}}\}.
\end{aligned}
$$

ALC Semantics

Definition 2.2. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *interpretation domain*, and a mapping $\cdot^{\mathcal{I}}$ that maps

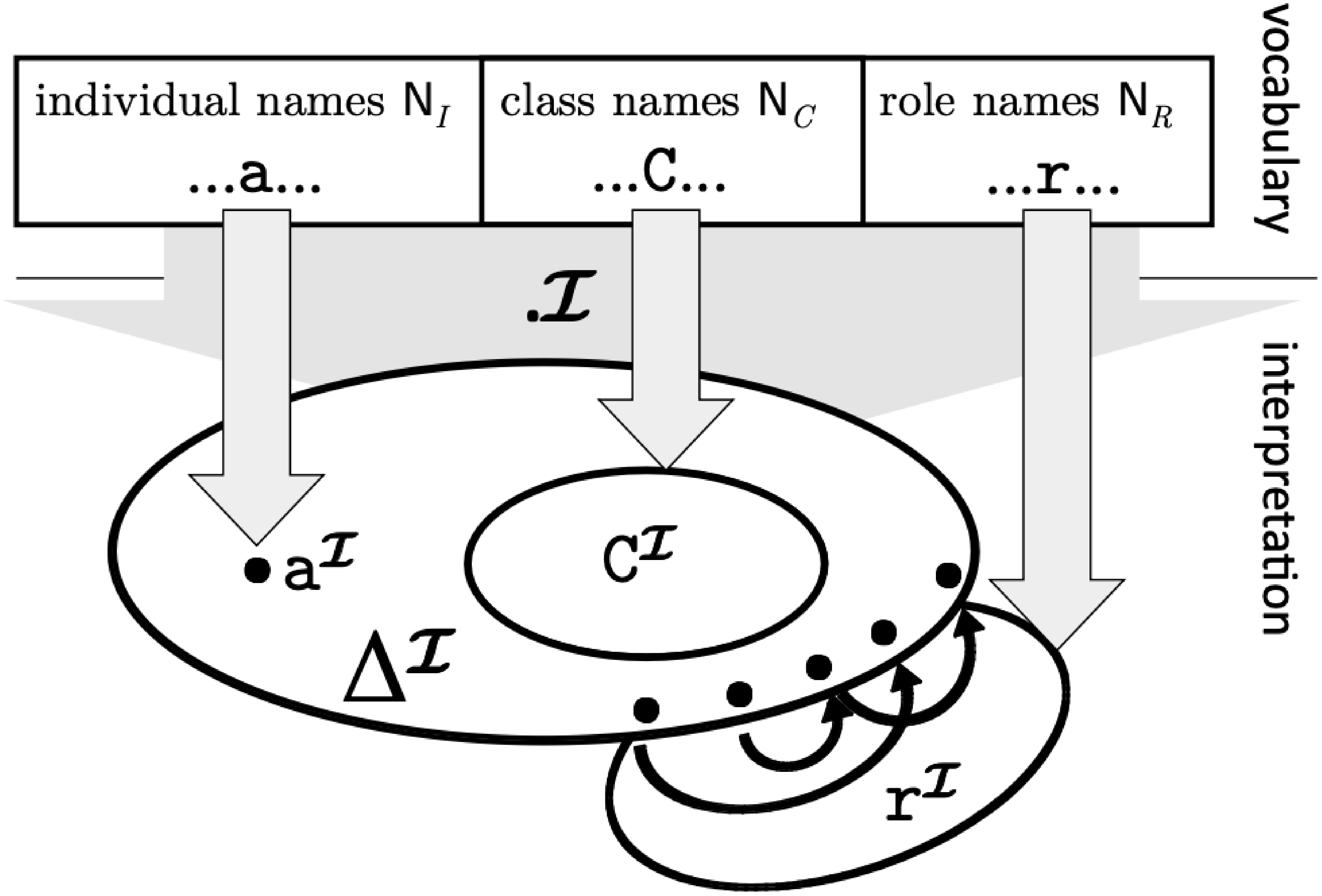
- every concept name $A \in \mathbf{C}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and
- every role name $r \in \mathbf{R}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
\boxed{(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}}&, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{there is an } e \in \Delta^{\mathcal{I}} \text{ with } (d, e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}, \\
(\forall r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}},\ \text{if } (d, e) \in r^{\mathcal{I}},\ \text{then } e \in C^{\mathcal{I}}\}.
\end{aligned}
$$

LHS: The interpretation I maps the conjunction of C and D to a set S.
RHS: The intersection S' of the set which I maps C to and the set which I maps D to.

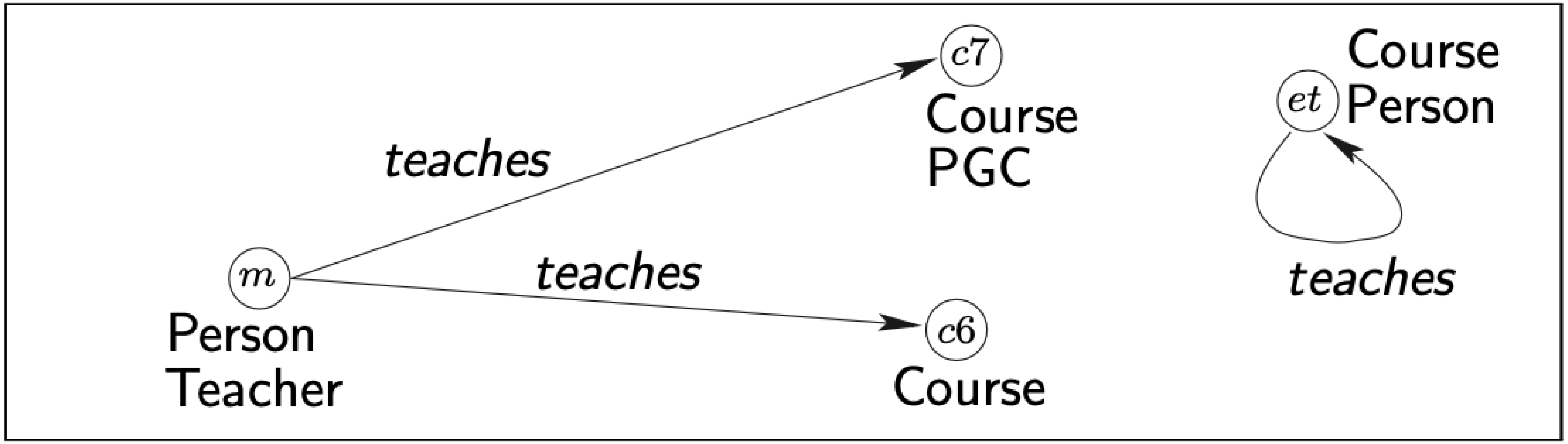
Equivalence: This statement asserts that S = S'.

Interpretations



Example: Interpretation

$$\begin{aligned}
\Delta^{\mathcal{I}} &= \{m, c6, c7, et\}, \\
\mathsf{Teacher}^{\mathcal{I}} &= \{m\}, \\
\mathsf{Course}^{\mathcal{I}} &= \{c6, c7, et\}, \\
\mathsf{Person}^{\mathcal{I}} &= \{m, et\}, \\
\mathsf{PGC}^{\mathcal{I}} &= \{c7\}, \\
\mathit{teaches}^{\mathcal{I}} &= \{(m, c6), (m, c7), (et, et)\}
\end{aligned}$$

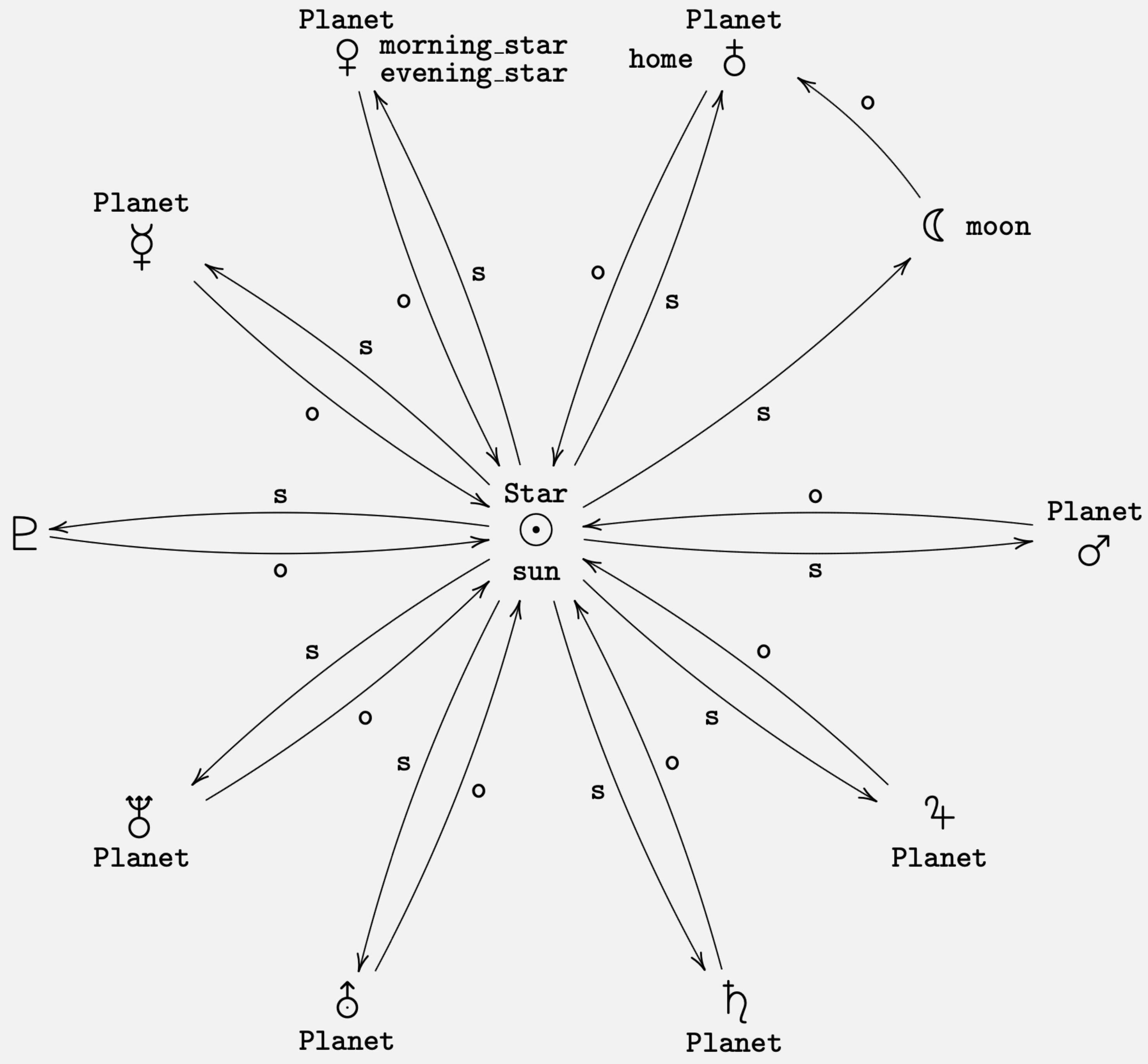


Exercise: Diagram a Model

- $\mathsf{N}_I = \{\mathtt{sun}, \mathtt{morning\_star}, \mathtt{evening\_star}, \mathtt{moon}, \mathtt{home}\}$.
- $\mathsf{N}_C = \{\mathtt{Planet}, \mathtt{Star}\}$.
- $\mathsf{N}_R = \{\mathtt{orbitsAround}, \mathtt{shinesOn}\}$.

We now define an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as follows: Let our domain $\Delta^{\mathcal{I}}$ contain the following elements: $\odot, \mercury, \venus, \earth, \leftmoon, \mars, \jupiter, \saturn, \uranus, \neptune, \pluto$. We define the interpretation function by

$$
\begin{aligned}
\mathtt{sun}^{\mathcal{I}} &= \odot \\
\mathtt{morning\_star}^{\mathcal{I}} &= \venus \\
\mathtt{evening\_star}^{\mathcal{I}} &= \venus \\
\mathtt{moon}^{\mathcal{I}} &= \leftmoon \\
\mathtt{home}^{\mathcal{I}} &= \earth
\end{aligned}
\qquad
\begin{aligned}
\mathtt{Planet}^{\mathcal{I}} &= \{\mercury, \venus, \earth, \mars, \jupiter, \saturn, \uranus, \neptune\} \\
\mathtt{Star}^{\mathcal{I}} &= \{\odot\} \\
\mathtt{orbitsAround}^{\mathcal{I}} &= \{\langle\mercury, \odot\rangle, \langle\venus, \odot\rangle, \langle\earth, \odot\rangle, \langle\mars, \odot\rangle, \langle\jupiter, \odot\rangle, \\
&\qquad \langle\saturn, \odot\rangle, \langle\uranus, \odot\rangle, \langle\neptune, \odot\rangle, \langle\pluto, \odot\rangle, \langle\leftmoon, \earth\rangle\} \\
\mathtt{shinesOn}^{\mathcal{I}} &= \{\langle\odot, \mercury\rangle, \langle\odot, \venus\rangle, \langle\odot, \earth\rangle, \langle\odot, \leftmoon\rangle, \langle\odot, \mars\rangle, \\
&\qquad \langle\odot, \jupiter\rangle, \langle\odot, \saturn\rangle, \langle\odot, \uranus\rangle, \langle\odot, \neptune\rangle, \langle\odot, \pluto\rangle\}
\end{aligned}
$$



Planet
morning_star
evening_star
Planet
home
o
Planet
moon
s
o
o
s
s
o
s
Star
sun
s
o
Planet
s
o
s
o
o
s
s
o
s
o
Planet
Planet
Planet
Planet

Outline

- A Brief History of Logics in Ontology Engineering
- Description Logic: ALC and Extensions
- The Bisimulation Theorem

Bisimulation

- Up to now, we've seen examples of differences in expressivity, e.g. ALC doesn't have a constructor for inverse roles.

- Examples are only suggestive, however; it isn't yet clear that, for example, a constructor for inverse roles cannot be defined using only the ALC syntax; if such a constructor can be defined, then ALCI is not more expressive than ALC

- We will be using a **bisimulation strategy** for demonstrating flavors of description logic are more or less expressive than one another

Bisimulation

- Examples are only suggestive, however; it isn't yet clear that, for example, a constructor for inverse roles cannot be defined using only the ALC syntax; if such a constructor can be defined, then ALCI is not more expressive than ALC

- **It is challenging to prove a negative**, i.e. "You cannot define an inverse role constructor in ALC"

- Bisimulation is a way to **prove a negative** via an indirect route

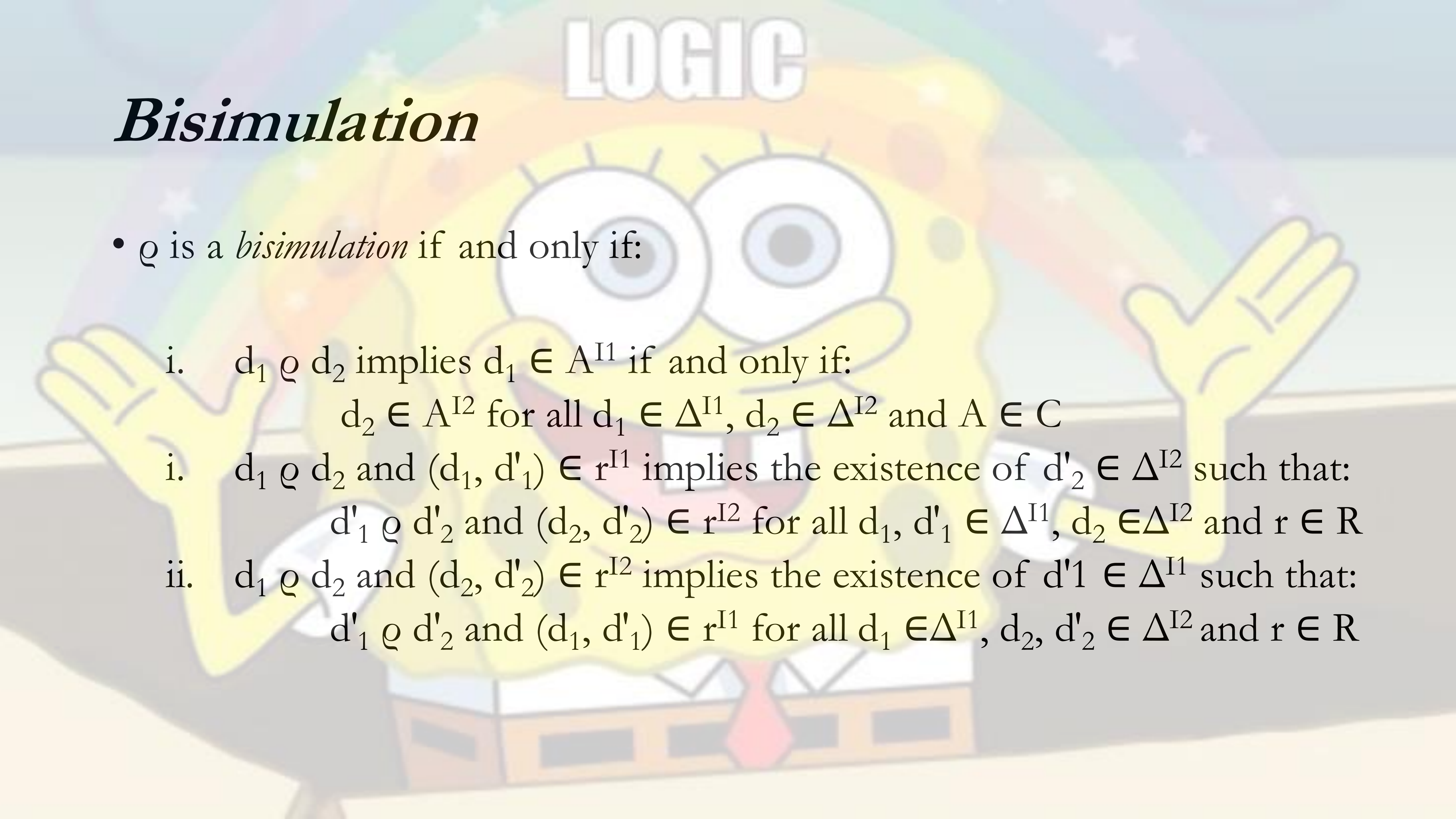
Bisimulation

- ϱ is a *bisimulation* if and only if:

    i. $d_1 \varrho d_2$ implies $d_1 \in A^{I1}$ if and only if:
    $d_2 \in A^{I2}$ for all $d_1 \in \Delta^{I1}$, $d_2 \in \Delta^{I2}$ and $A \in C$

    i. $d_1 \varrho d_2$ and $(d_1, d'_1) \in r^{I1}$ implies the existence of $d'_2 \in \Delta^{I2}$ such that:
    $d'_1 \varrho d'_2$ and $(d_2, d'_2) \in r^{I2}$ for all $d_1, d'_1 \in \Delta^{I1}$, $d_2 \in \Delta^{I2}$ and $r \in R$

    ii. $d_1 \varrho d_2$ and $(d_2, d'_2) \in r^{I2}$ implies the existence of $d'1 \in \Delta^{I1}$ such that:
    $d'_1 \varrho d'_2$ and $(d_1, d'_1) \in r^{I1}$ for all $d_1 \in \Delta^{I1}$, $d_2, d'_2 \in \Delta^{I2}$ and $r \in R$



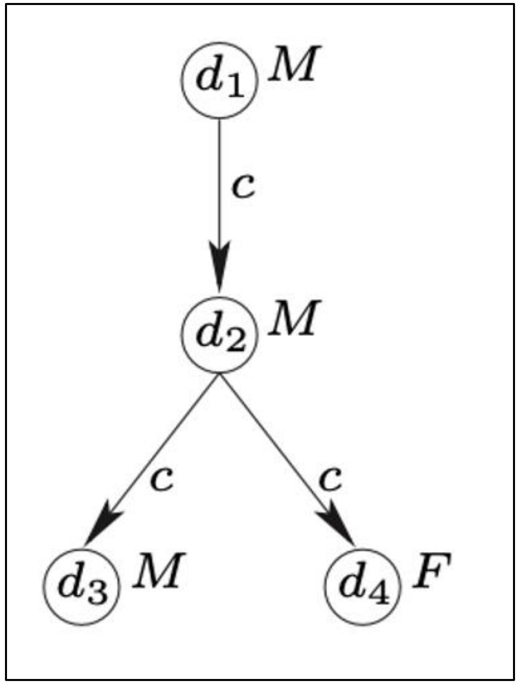
Bisimulation

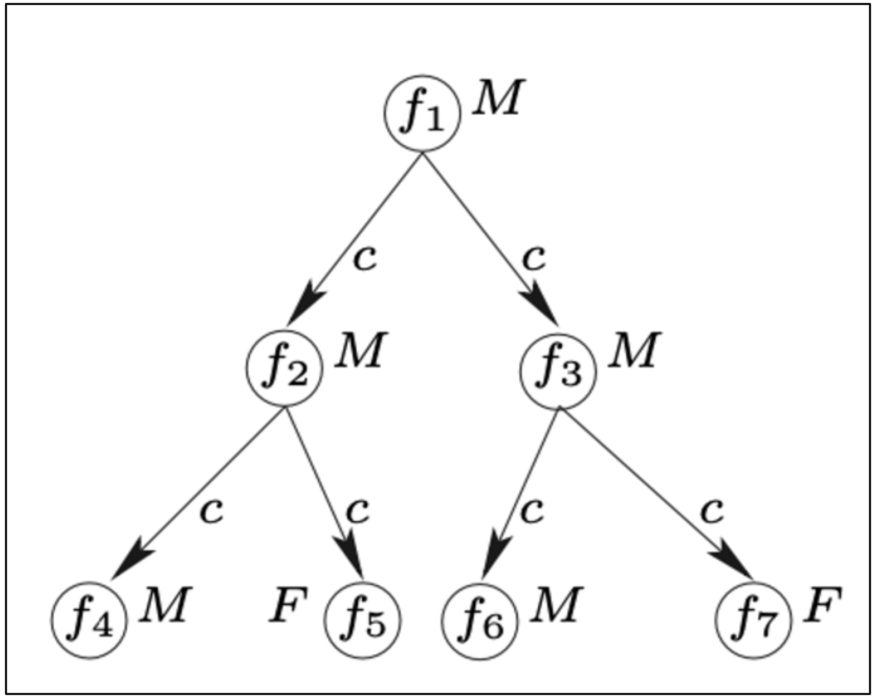
- ϱ is a *bisimulation* if and only if:

  i. $d_1 \ \varrho \ d_2$ implies $d_1 \in A^{I1}$ if and only if:
  $d_2 \in A^{I2}$ for all $d_1 \in \Delta^{I1}$, $d_2 \in \Delta^{I2}$ and $A \in C$

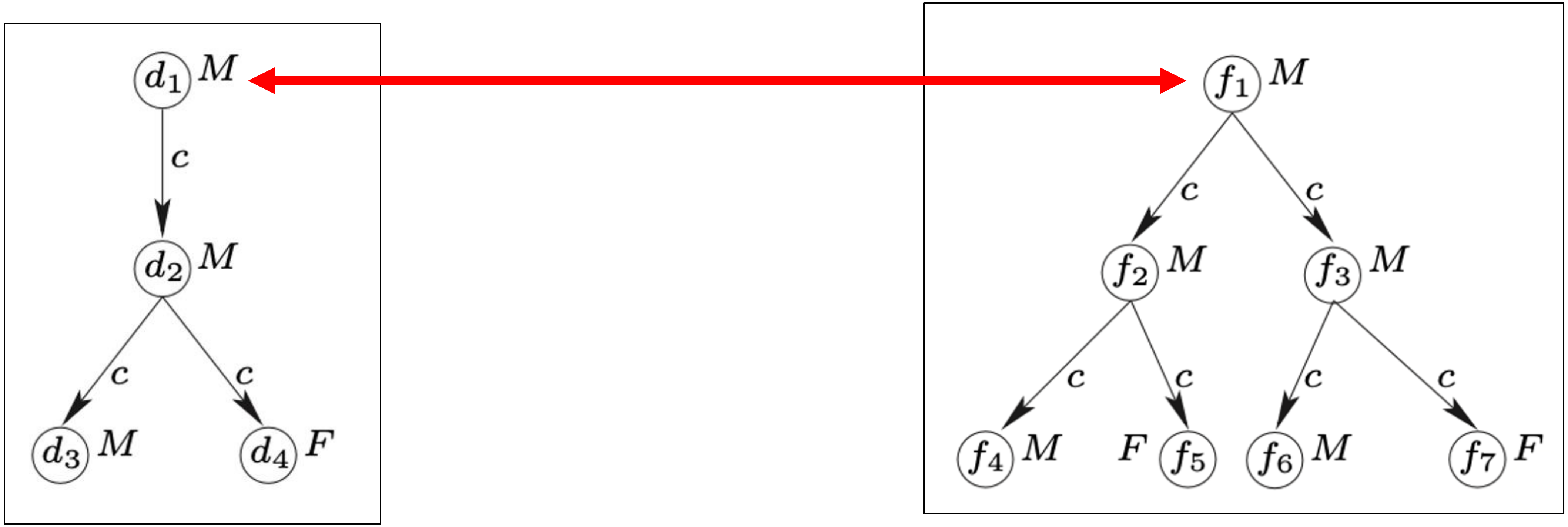
  i. $d_1 \ \varrho \ d_2$ and $(d_1, d'_1) \in r^{I1}$ implies the existence of $d'_2 \in \Delta^{I2}$ such that:
  $d'_1 \ \varrho \ d'_2$ and $(d_2, d'_2) \in r^{I2}$ for all $d_1, d'_1 \in \Delta^{I1}$, $d_2 \in \Delta^{I2}$ and $r \in R$

  ii. $d_1 \ \varrho \ d_2$ and $(d_2, d'_2) \in r^{I2}$ implies the existence of $d'1 \in \Delta^{I1}$ such that:
  $d'_1 \ \varrho \ d'_2$ and $(d_1, d'_1) \in r^{I1}$ for all $d_1 \in \Delta^{I1}$, $d_2, d'_2 \in \Delta^{I2}$ and $r \in R$



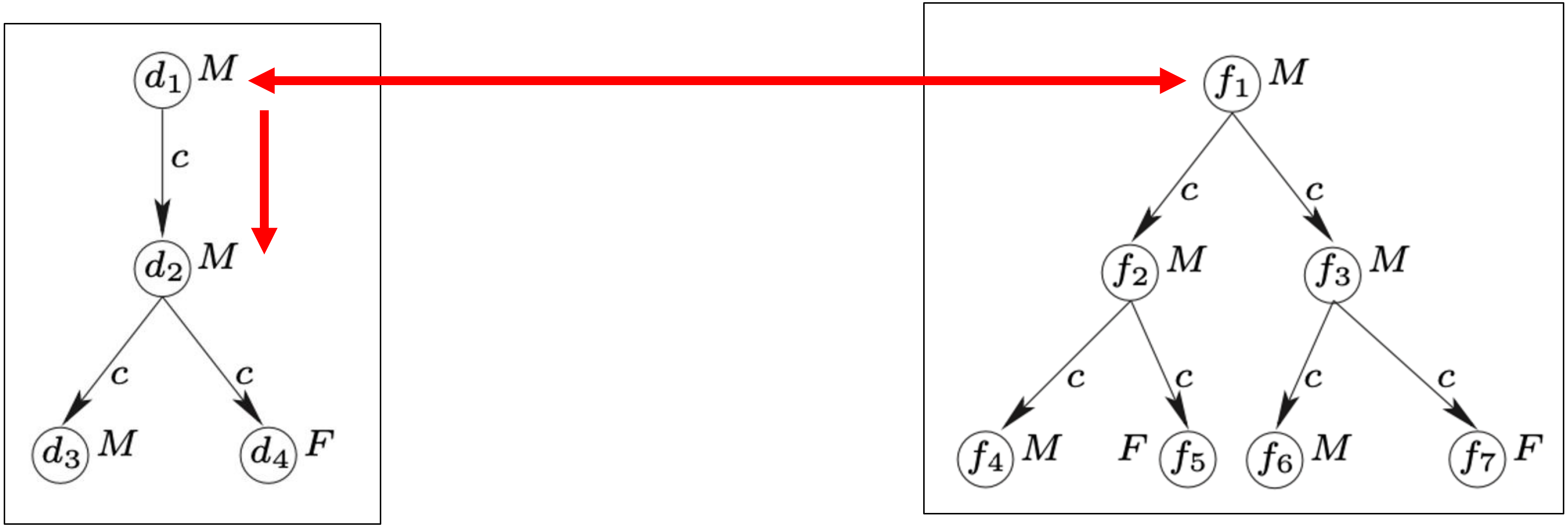


- **Claim:** d_1 is bisimilar to f_1



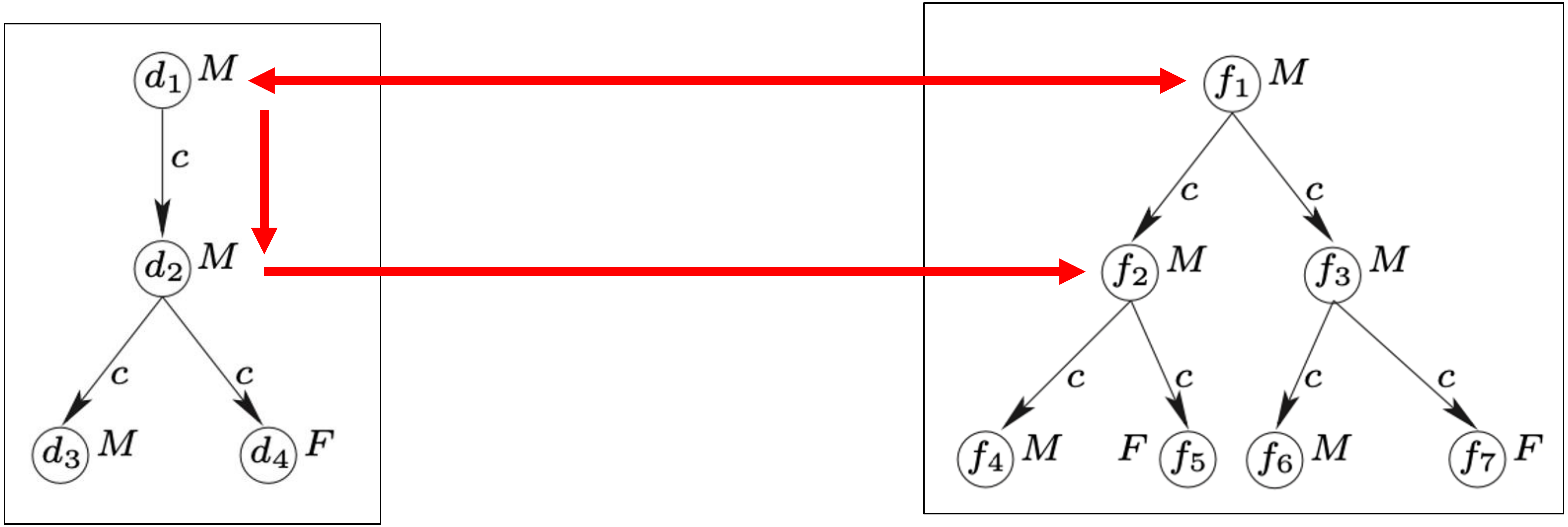
- **Claim:** d_1 is bisimilar to f_1

- **Invariance:** d_1 and f_1 are both instances of M



- **Claim:** d_1 is bisimilar to f_1

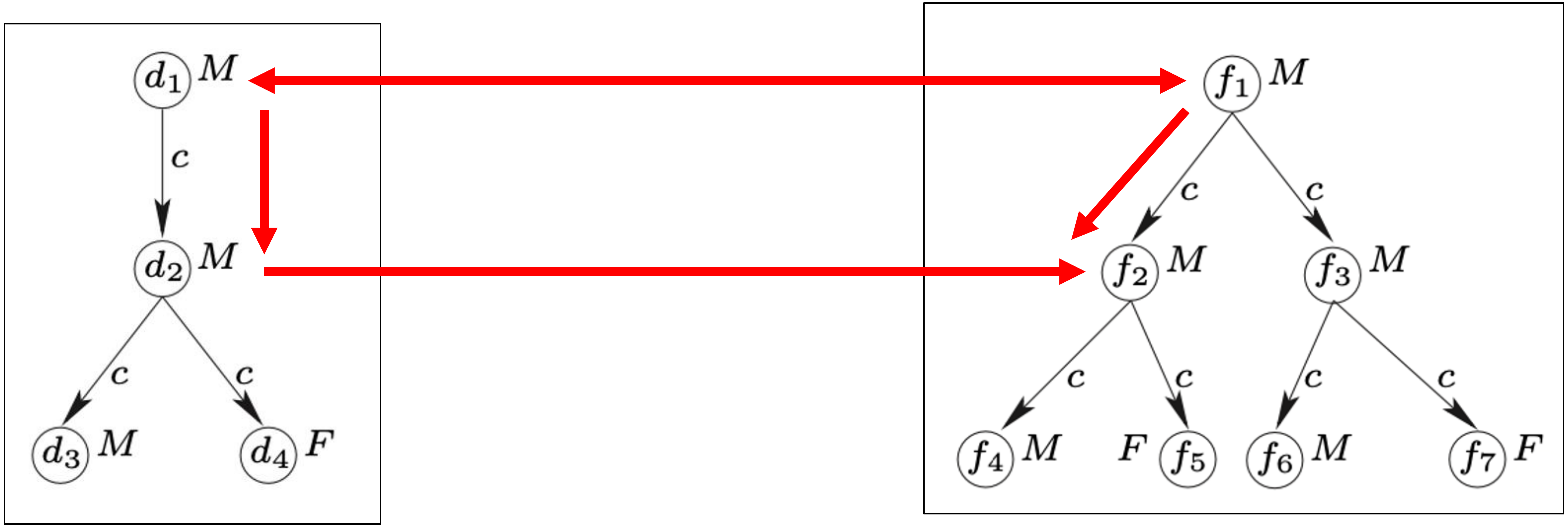
- **Zig:**
  If role c relates d_1 to d_2



- **Claim:** d_1 is bisimilar to f_1

- **Zig:**

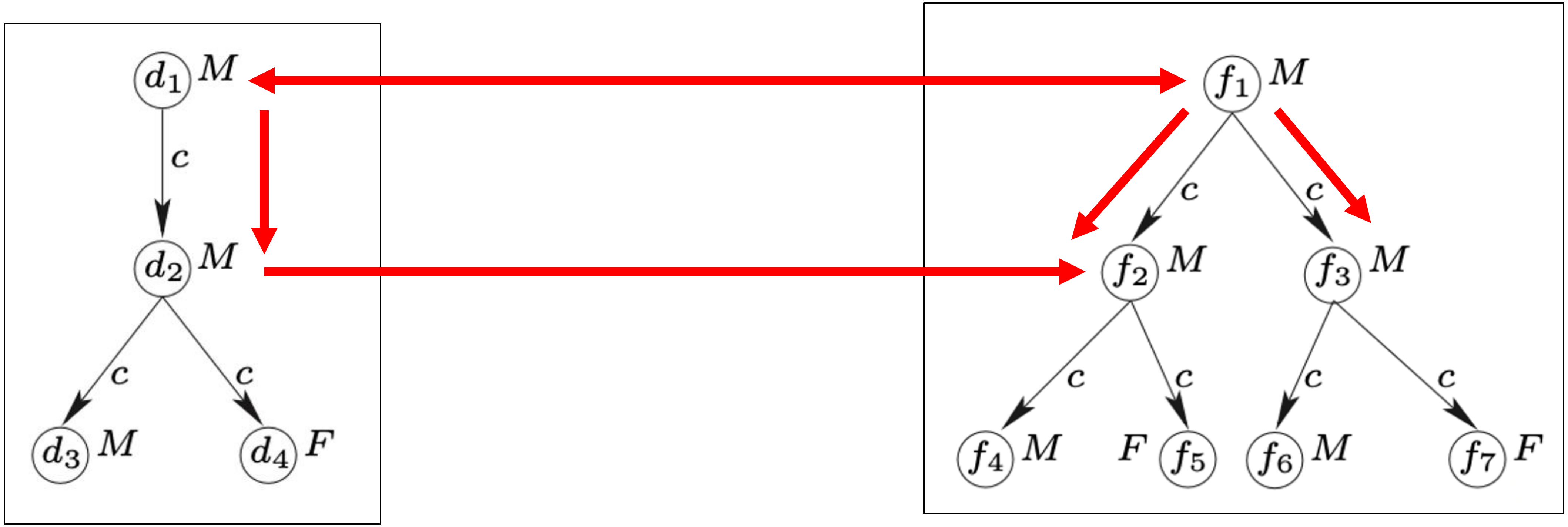
  If role c relates d_1 to d_2 then there is a mapping from d_2 to f_2 where d_2 and f_2 are both instances of M



- **Claim:** d_1 is bisimilar to f_1

- **Zig:**

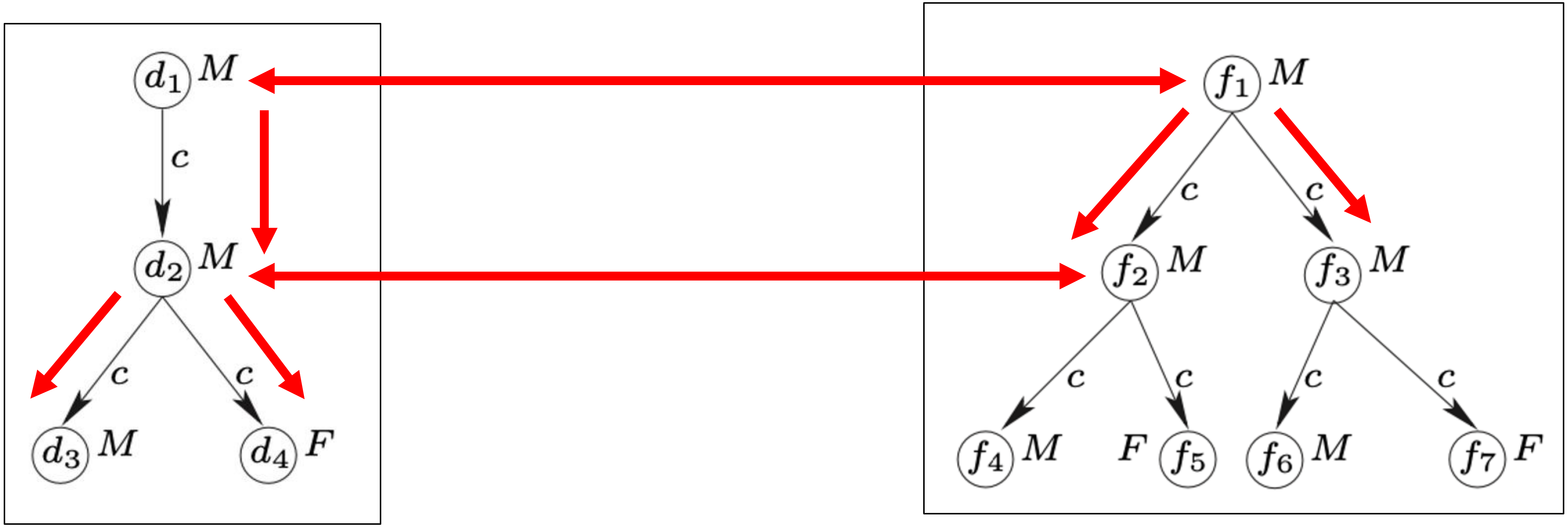
  If role c relates d_1 to d_2 then there is a mapping from d_2 to f_2 where d_2 and f_2 are both instances of M and role c maps f_1 to f_2



- **Claim:** d_1 is bisimilar to f_1

- **Zig:**

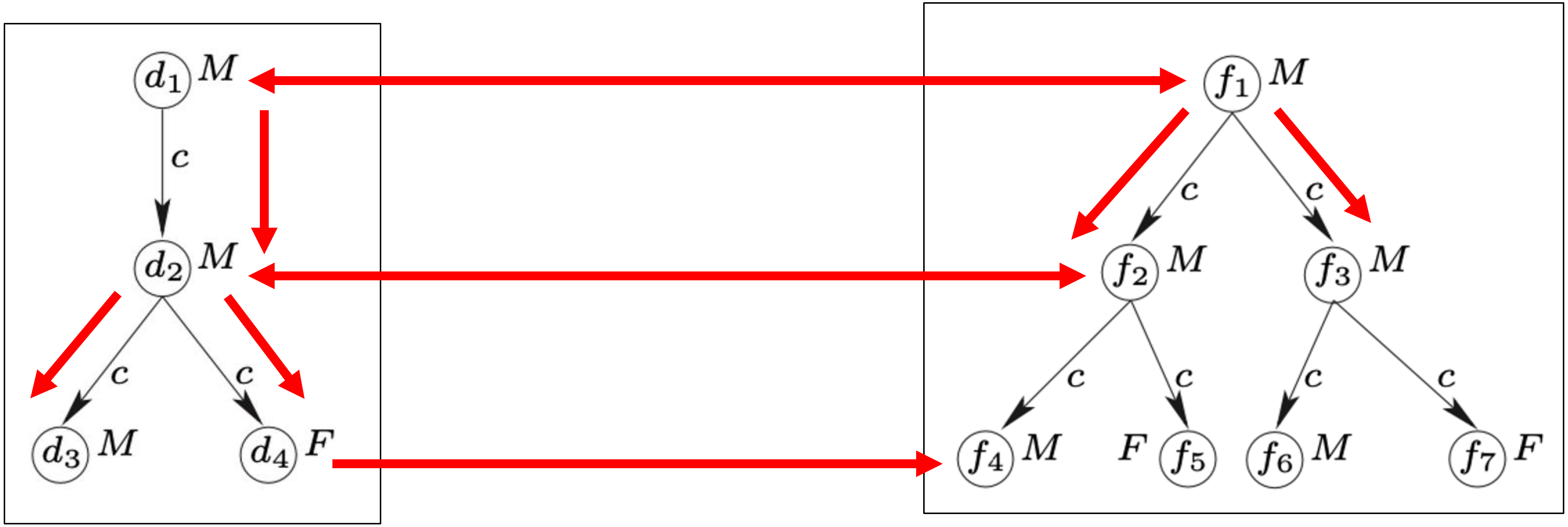
  If role c relates d_1 to d_2 then there is a mapping from d_2 to f_2 where d_2 and f_2 are both instances of M and role c maps f_1 to f_2 and f_1 to f_3



- **Claim:** d_1 is bisimilar to f_1

- **Zag:**

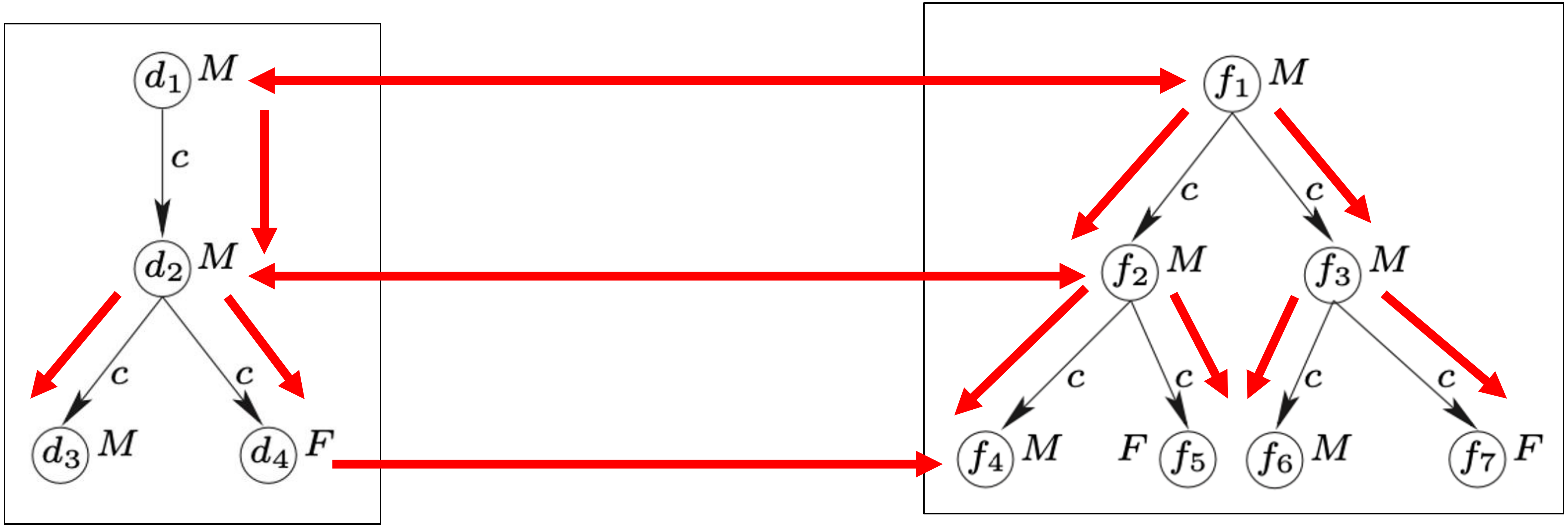
  If role c relates d_2 to d_3 and d_4



- **Claim:** d_1 is bisimilar to f_1

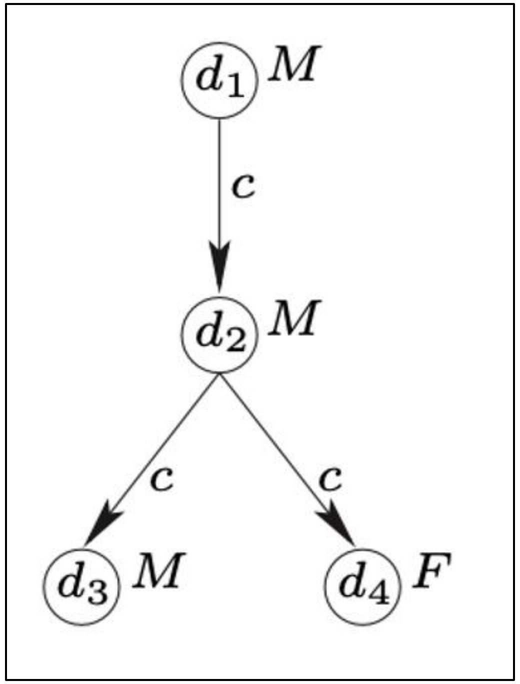
- **Zag:**

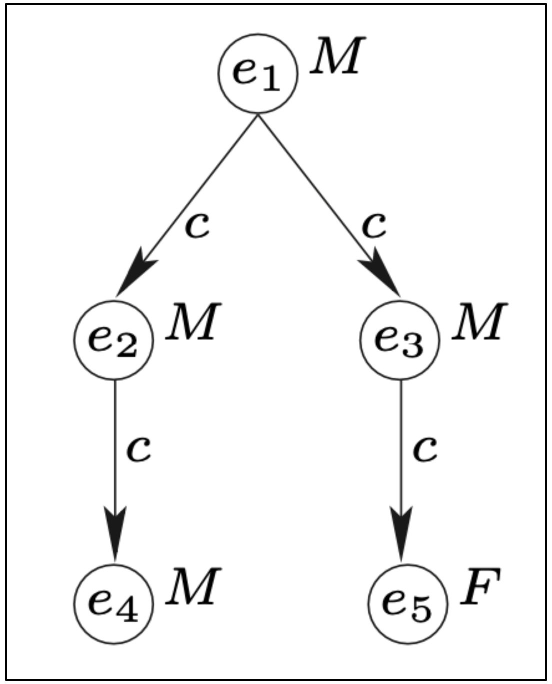
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to f_4 and d_4 to f_5 as well as from d_3 to f_6 and d_4 to f_7 where d_3, f_4, and f_6 are instances of M and d_4, f_5, and f_7 are instances of F and



- **Claim:** d_1 is bisimilar to f_1

- **Zag:**
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to f_4 and d_4 to f_5 as well as from d_3 to f_6 and d_4 to f_7 where d_3, f_4, and f_6 are instances of M and d_4, f_5, and f_7 are instances of F and c relates f_2 to f_4 and f_5 and related f_3 to f_6 and f_7





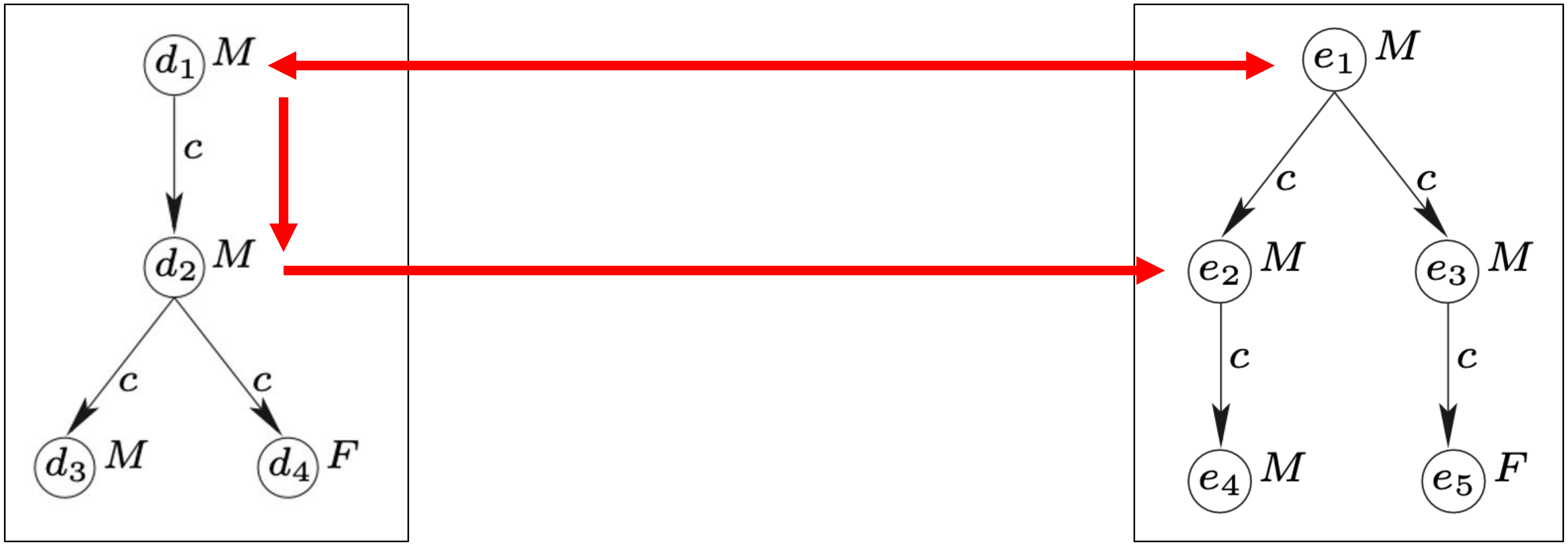
- **Claim:** d_1 is not bisimilar to e_1

- **Claim:** d_1 is not bisimilar to e_1
- **Invariance**: d_1 and e_1 are both instances of M

- **Claim:** d_1 is not bisimilar to e_1

- **Zig:**

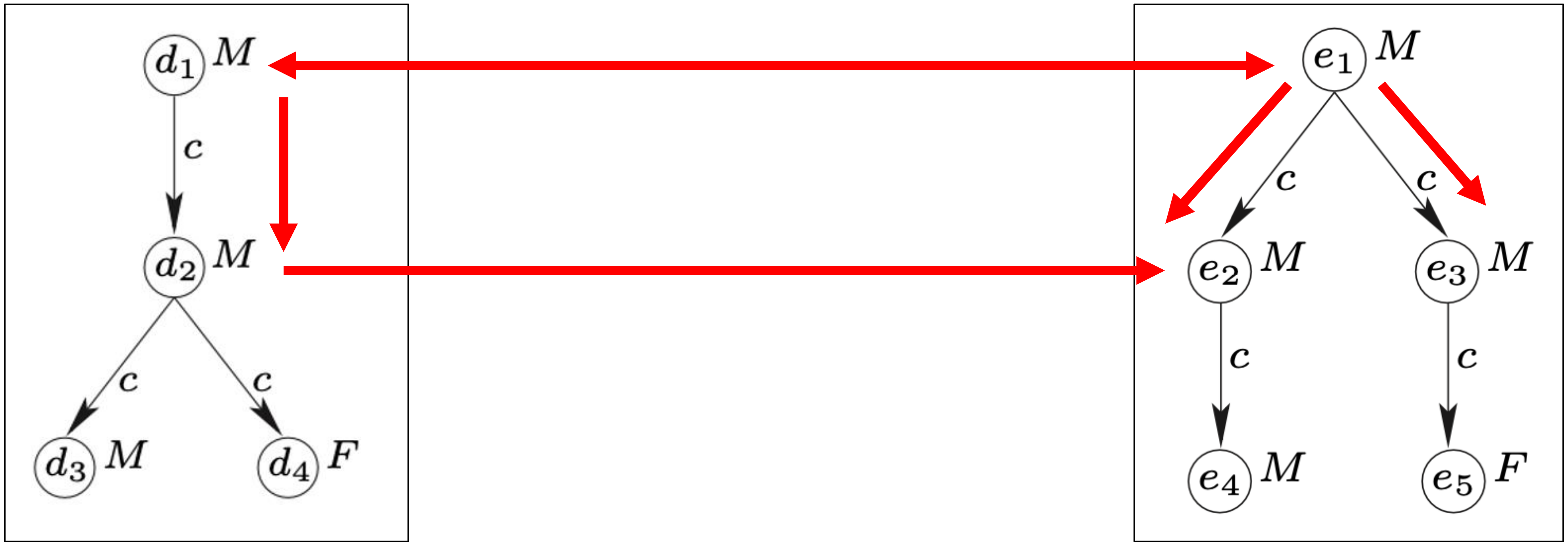
  If role c relates d_1 to d_2



- **Claim:** d_1 is not bisimilar to e_1

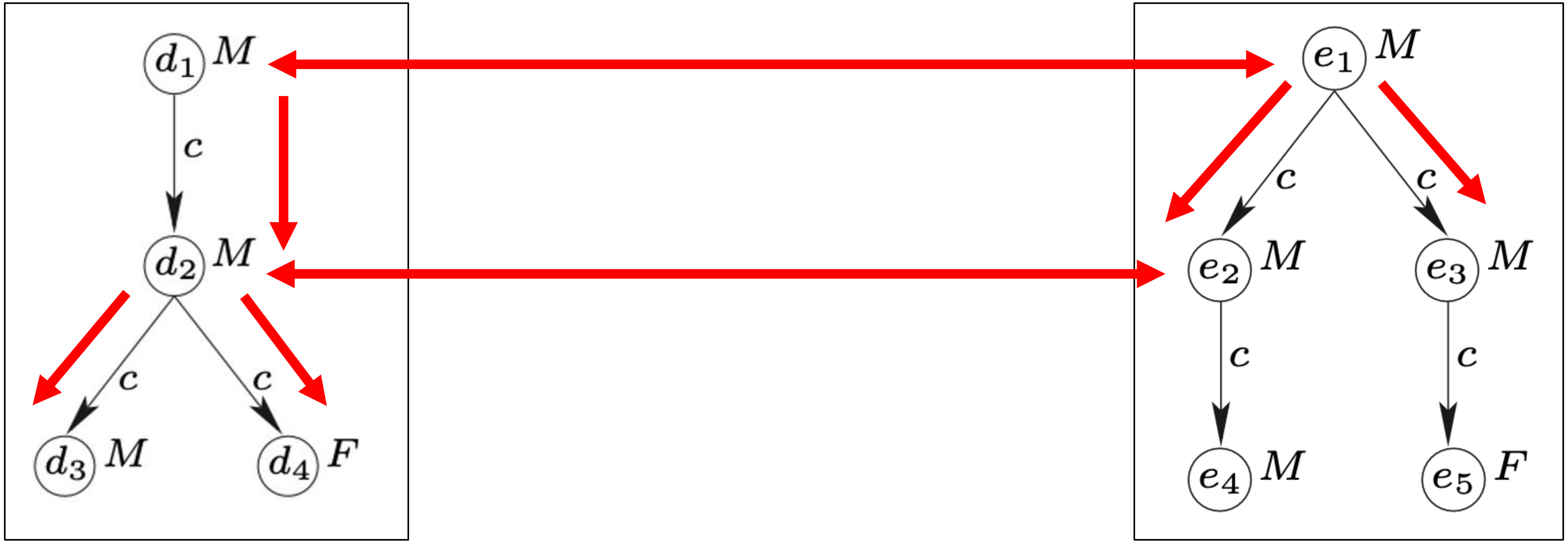
- **Zig:**

  If role c relates d_1 to d_2 then there is a mapping from d_2 to e_2 where d_2 and e_2 are both instances of M and from d_2 to e_3 where d_2 and e_3 are both instances of M



- **Claim:** d_1 is not bisimilar to e_1

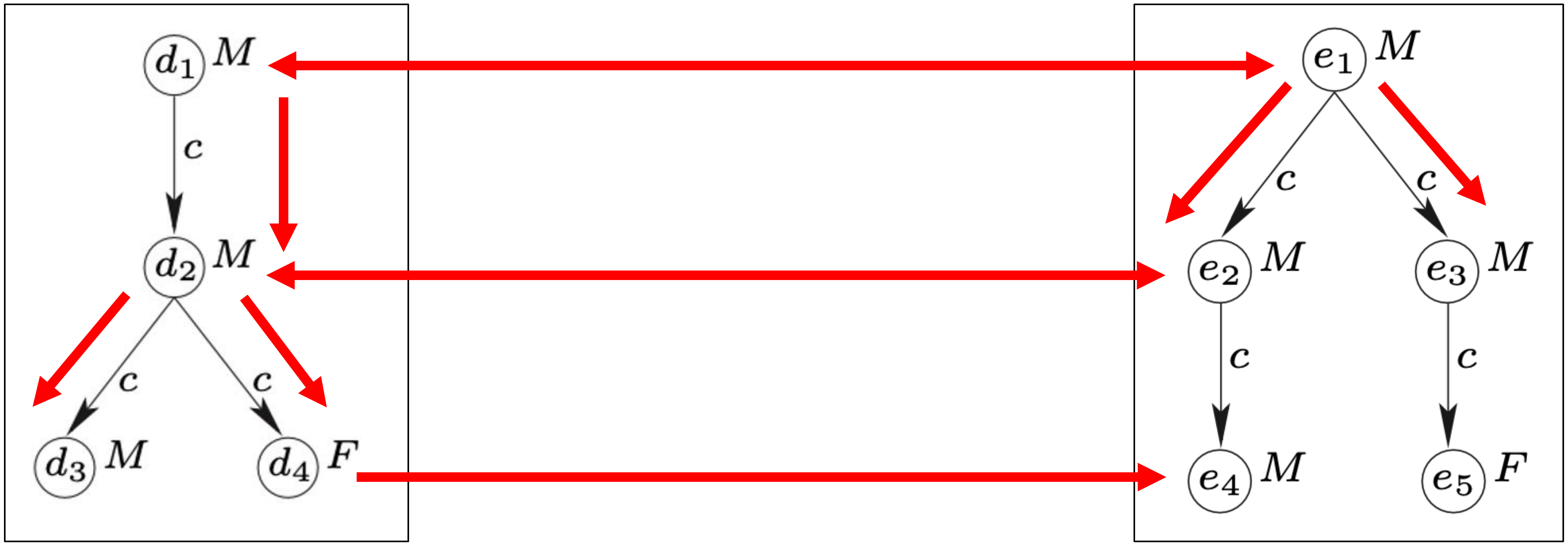
- **Zig:**
  If role c relates d_1 to d_2 then there is a mapping from d_2 to e_2 where d_2 and e_2 are both instances of M and from d_2 to e_3 where d_2 and e_3 are both instances of M and role c maps e_1 to e_2 and e_1 to e_3



- **Claim:** d_1 is not bisimilar to e_1

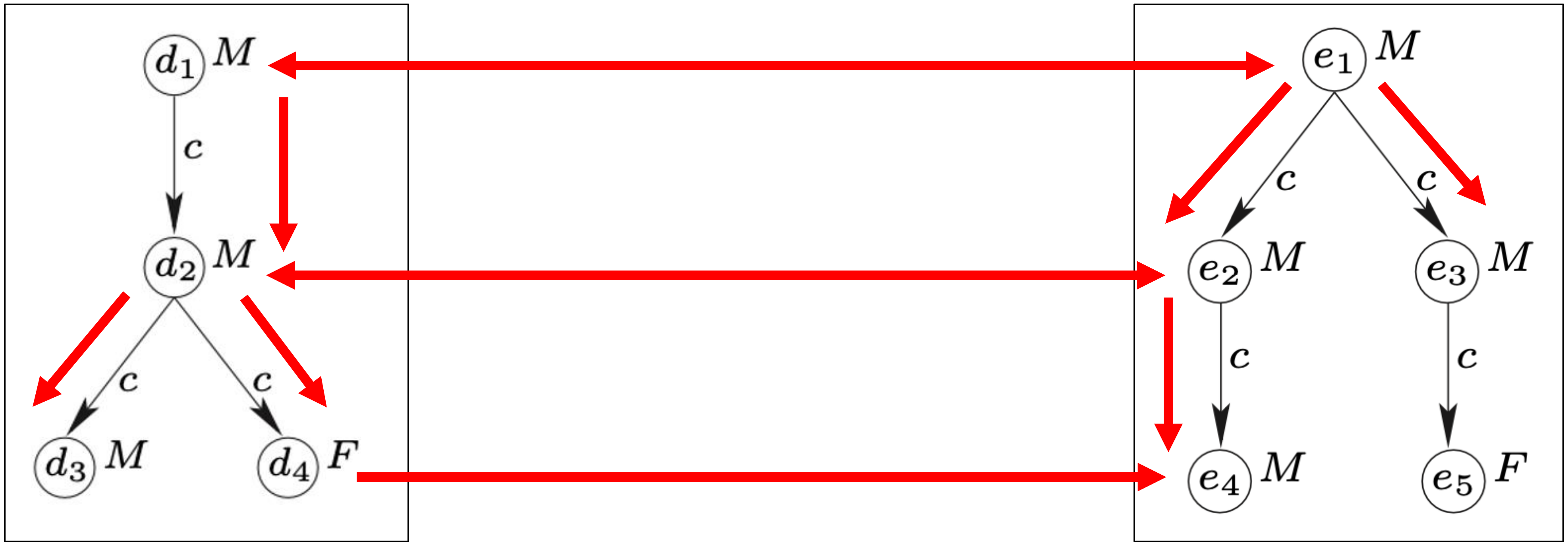
- **Zag:**

  If role c relates d_2 to d_3 and d_4



- **Claim:** d_1 is not bisimilar to e_1

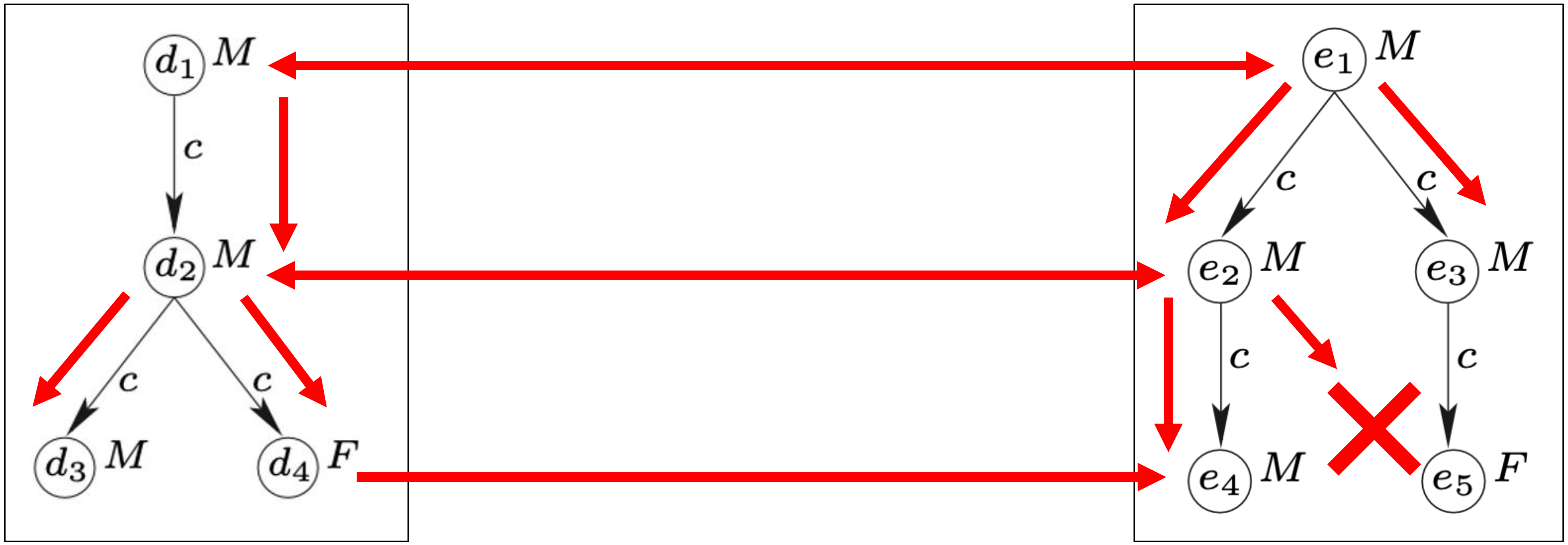
- **Zag:**
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

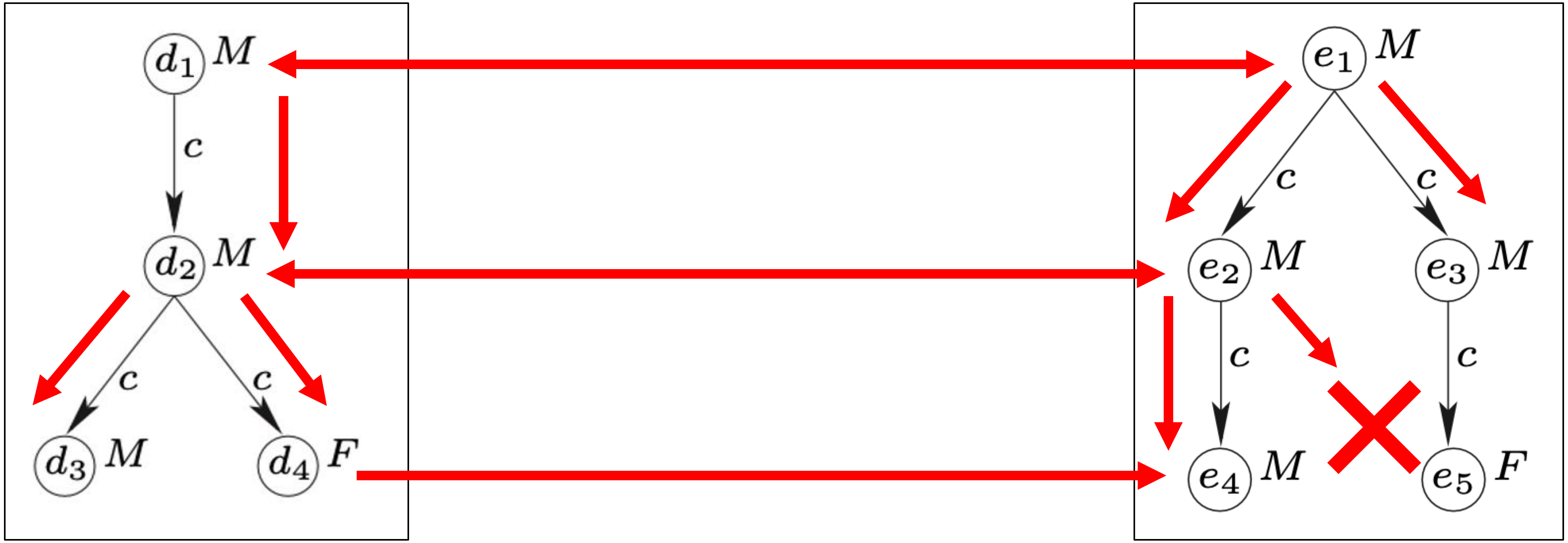
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F and c relates e_2 to e_4



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F and c relates e_2 to e_4 and c relates e_2 to e_5



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F and c relates e_2 to e_4 and ~~c relates e_2 to e_5~~

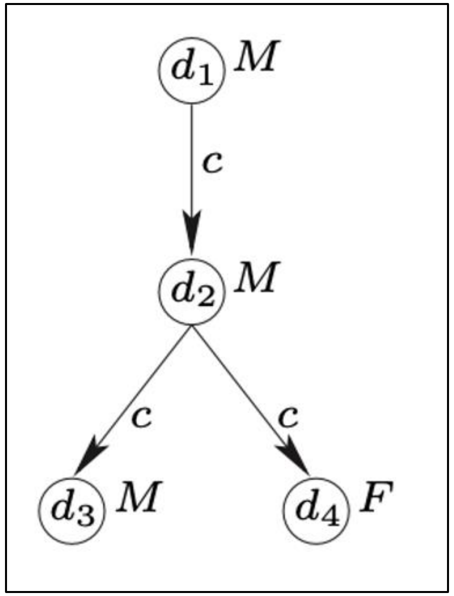
Expressivity

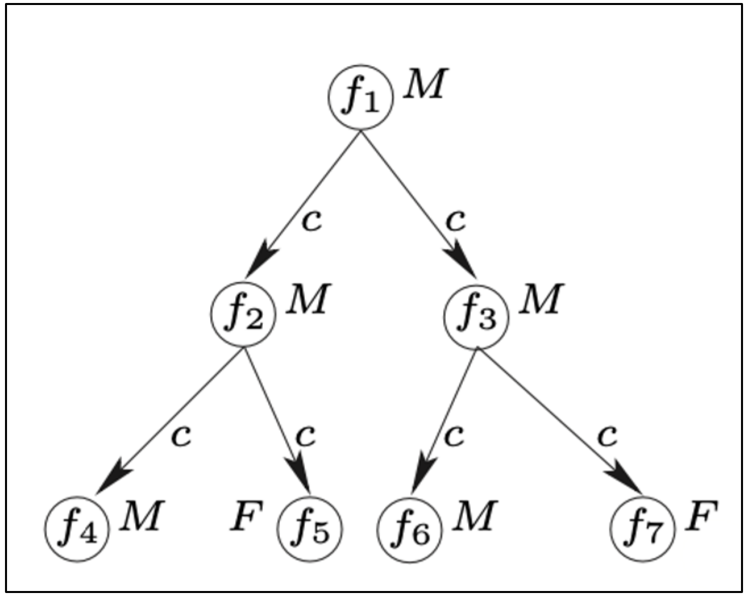
- The language of ALC is not expressive enough to distinguish between bisimilar elements.
- Bisimulation is a relationship between interpretations/elements; interpretations are distinct from syntaxes
- For example, bisimulations between interpretations are distinct from the syntax of ALC

Expressivity

$$\exists c.(M \sqcap \exists c.M \sqcap \exists c.F)$$

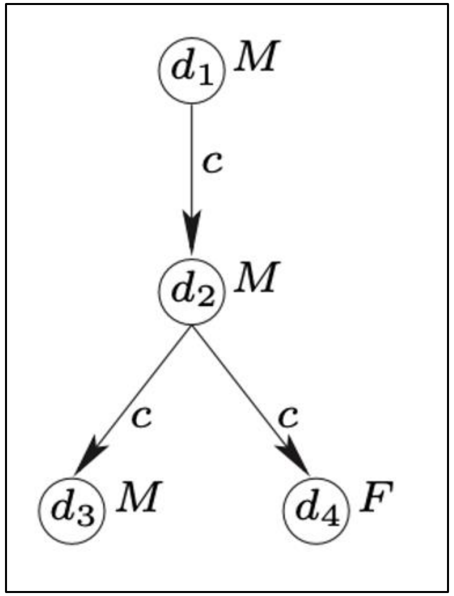
$$\exists y(c(x,y) \ \& \ M(y) \ \& \ \exists z(c(y,z) \ \& \ M(z)) \ \& \ \exists u(c(y,u) \ \& \ F(u)))$$

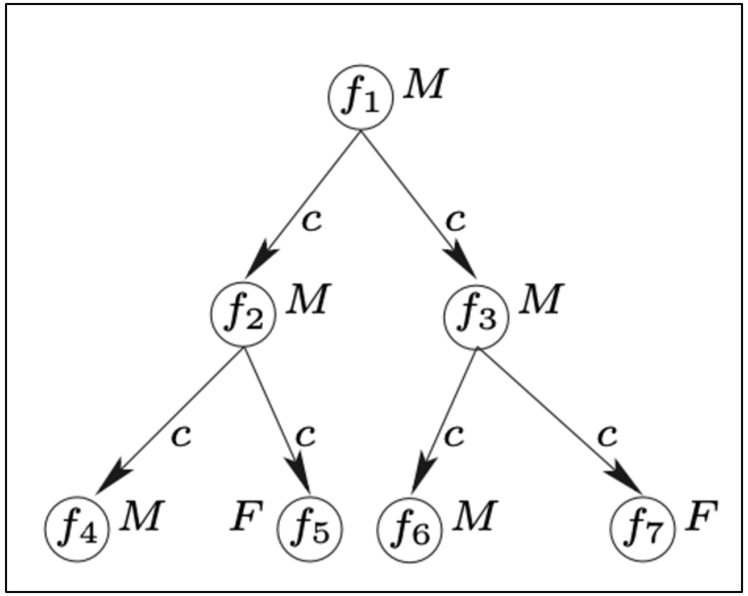




Expressivity

Any x that has *at least one* son y who has *at least one* son z and *at least one* one daughter u

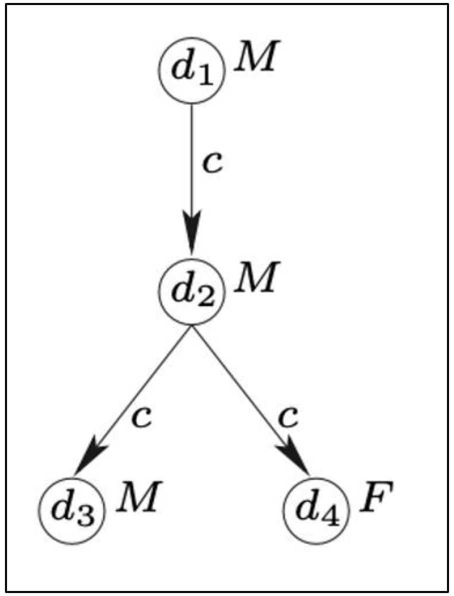


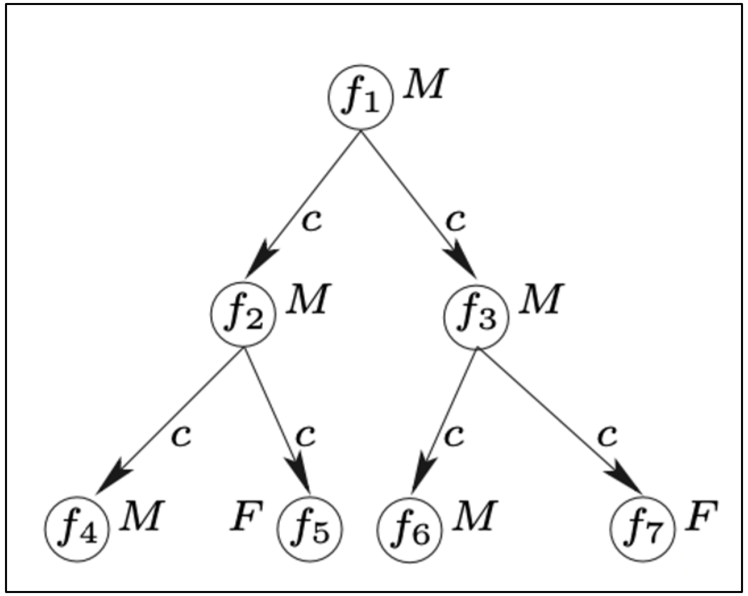


Expressivity

Any x that has *at least one* son y who has *at least one* son z and *at least one* one daughter u

Both graphs satisfy the ALC expression: ∃c.(M ⊓ ∃c.M ⊓ ∃c.F)





Expressivity

- There is an *ALCI* concept C such that $C \neq D$ holds for all ALC concepts D.

- Recall, ALCI adds only "$\exists r^-.\top$" to the syntax of ALC. To prove ALCI is more expressive than ALC, we must show there is no expression in ALC that is equivalent to $\exists r^-.\top$

- Suppose there is such an expression in ALC – call it D - we will show this assumption leads to contradiction.

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:

- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

ALCI > ALC

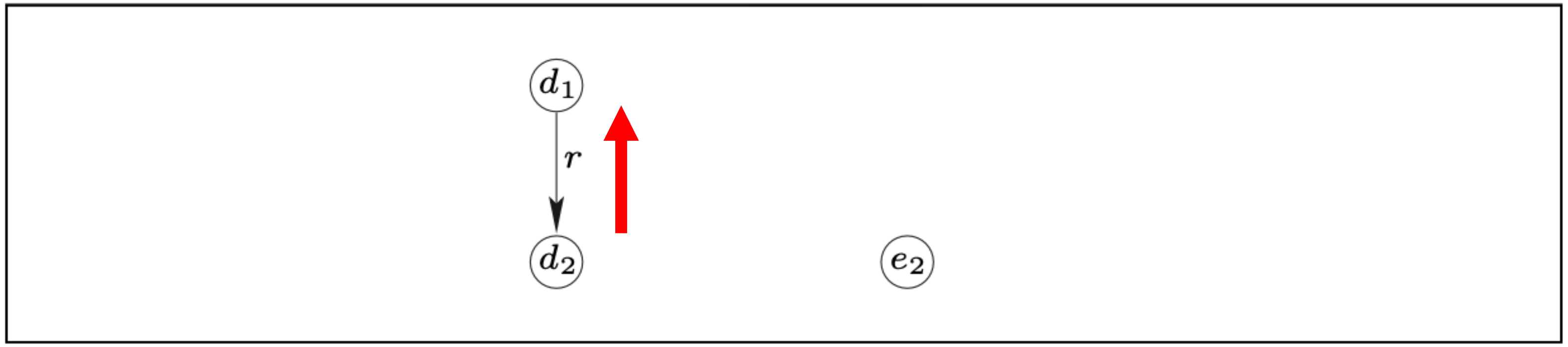
- Consider d_2 and e_2 in the following diagram:

- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

- However, $d_2 \in (\exists r-.\top)^{I1}$

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:

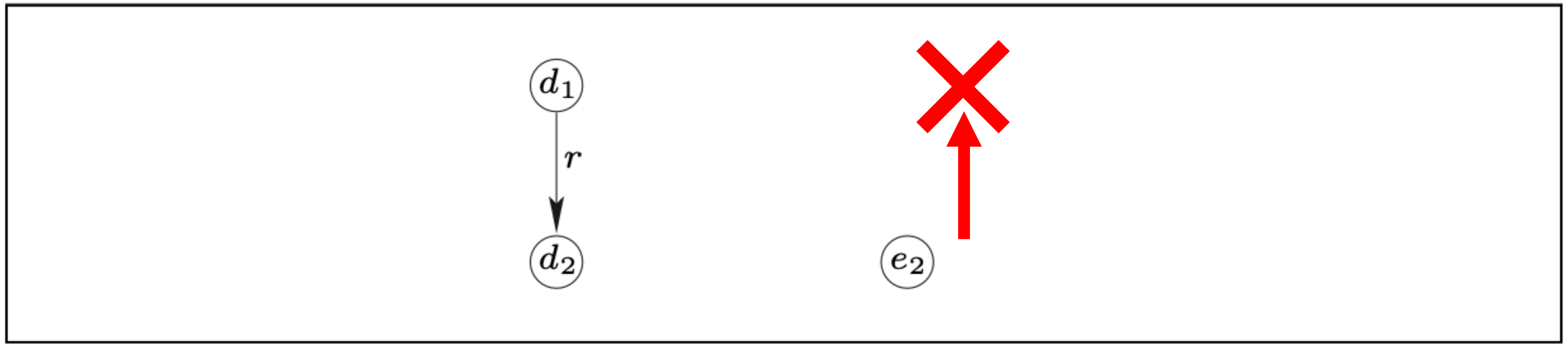


- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

- However, $d_2 \in (\exists r-.\top)^{I1}$

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:



- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

- However, $d_2 \in (\exists r-.\top)^{I1}$ and $e_2 \notin (\exists r-.\top)^{I2}$

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:

- That is, the ALCI expression $\exists r^{-}.\top$ can be satisfied by d_2 in the left graph but not e_2 in the right, since the latter lacks any role to have an inverse

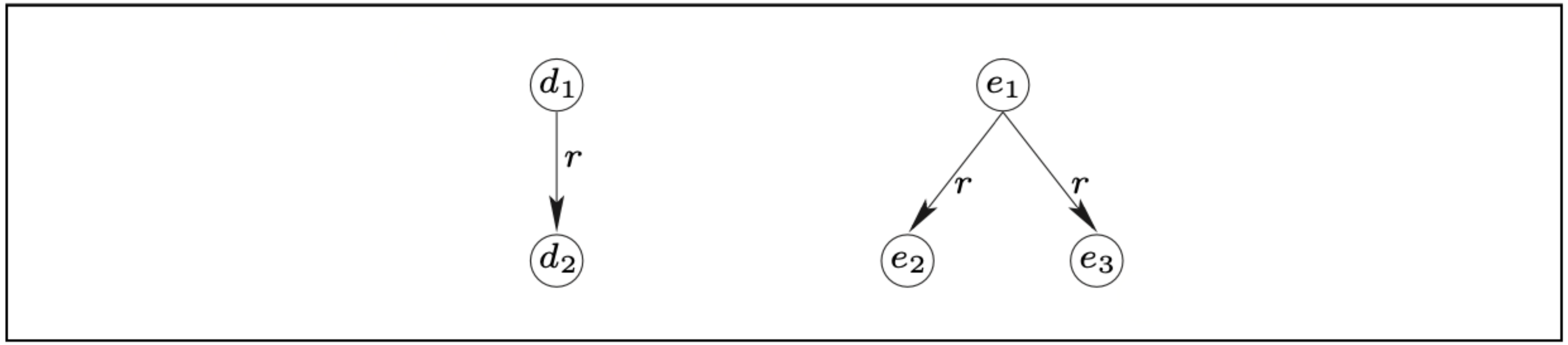
- Because there is a bisimulation between d_2 and e_2 and d_2 satisfies $\exists r^{-}.\top$ but e_2 doesn't, *ALCI can distinguish between bisimilar graphs that ALC cannot*

Expressivity

- There is an *ALCN* concept C such that C≠D holds for all ALC concepts D.
- Recall, ALCN adds only "≤r.1" to the syntax of ALC. To prove ALCI is more expressive than ALC, we must show there is no expression in ALC that is equivalent to ≤r.1
- Suppose there is such an expression in ALC – call it D - we will show this assumption leads to contradiction.

ALCN > ALC

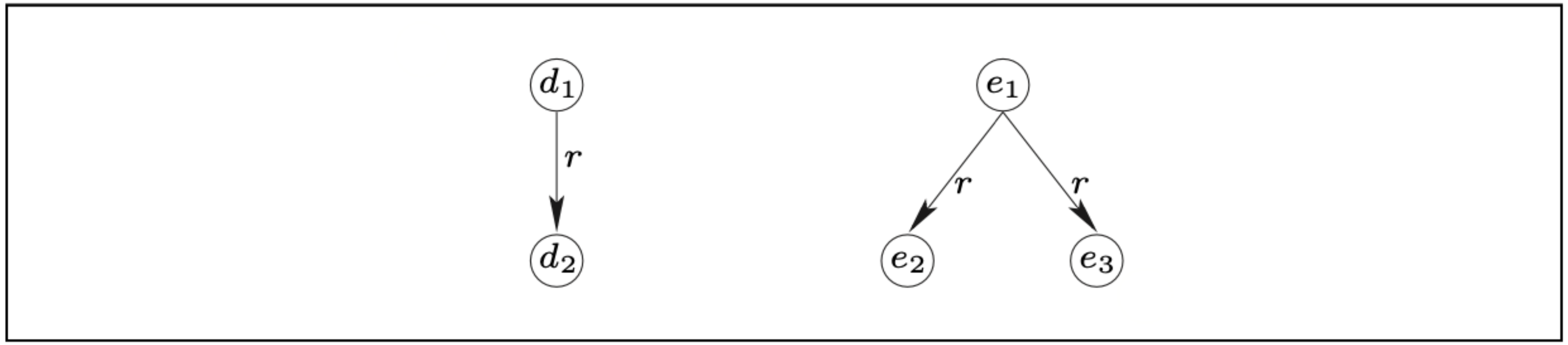
- Consider the following diagram:



- There is a bisimulation between the elements, so $d_1 \in D^{I1}$ just in case $e_1 \in D^{I2}$

ALCN > ALC

- Consider the following diagram:

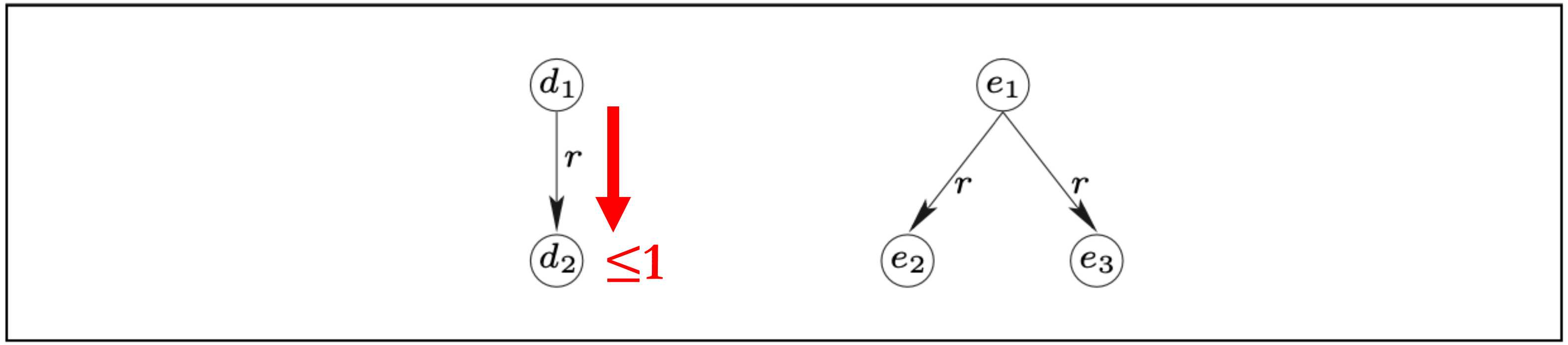


- There is a bisimulation between the elements, so $d_1 \in D^{I1}$ just in case $e_1 \in D^{I2}$

- However, $d_1 \in (\leq r.1)^{I1}$

ALCN > ALC

- Consider the following diagram:

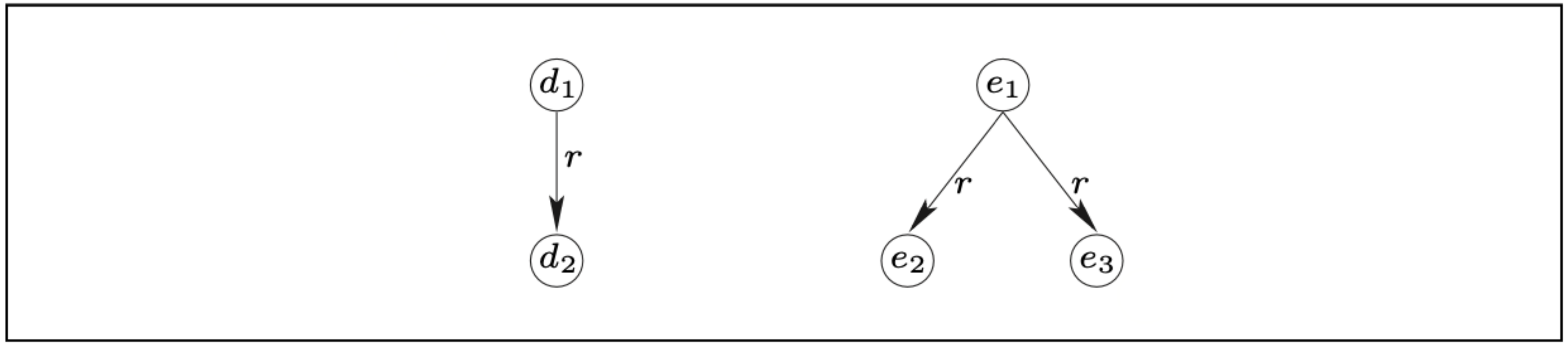


- There is a bisimulation between the elements, so $d_1 \in D^{I1}$ just in case $e_1 \in D^{I2}$

- However, $d_1 \in (\leq r.1)^{I1}$

ALCN > ALC

- Consider the following diagram:

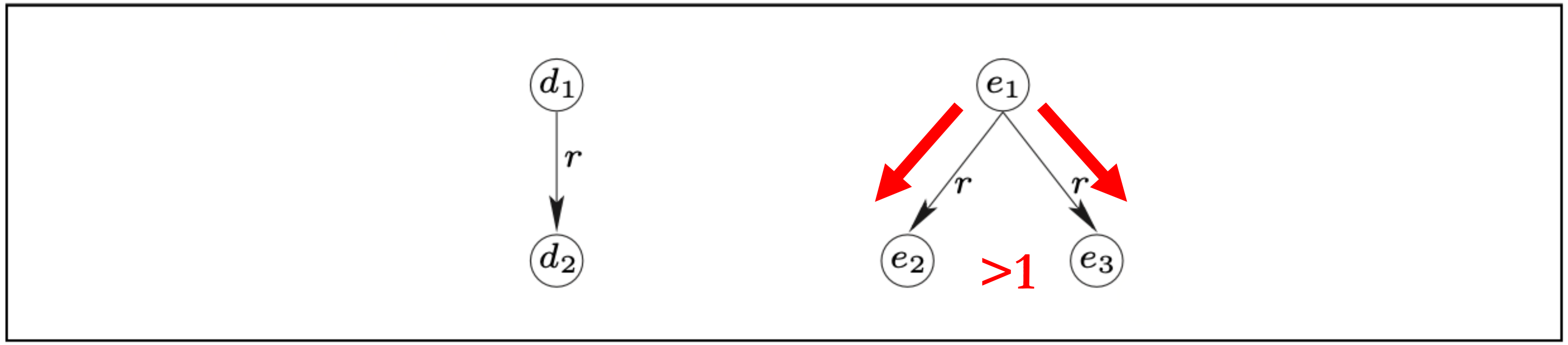


- There is a bisimulation between the elements, so $d_1 \in D^{I1}$ just in case $e_1 \in D^{I2}$

- However, $d_1 \in (\leq r.1)^{I1}$ and $e_1 \notin (\leq r.1)^{I2}$

ALCN > ALC

- Consider the following diagram:

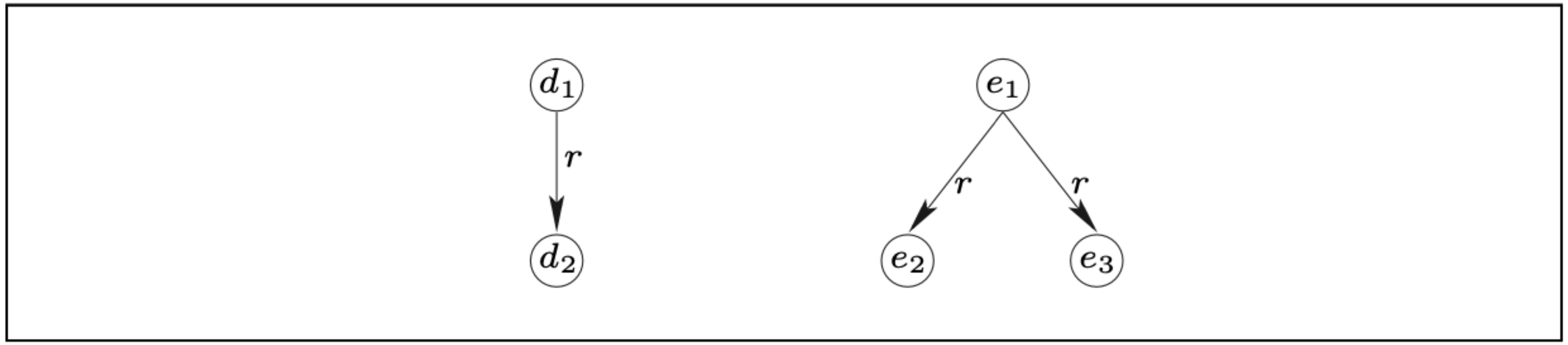


- There is a bisimulation between the elements, so $d_1 \in D^{I1}$ just in case $e_1 \in D^{I2}$

- However, $d_1 \in (\leq r.1)^{I1}$ and $e_1 \notin (\leq r.1)^{I2}$

ALCN > ALC

- Consider the following diagram:



- That is, the ALCN expression $\leq$r.1 can be satisfied by d_1 in the left graph but not e_1 in the right, since the latter is related to more than 1 element

- Because there is a bisimulation between d_1 and e_1 and d_1 satisfies $\leq$r.1 but e_1 doesn't, *ALCN can distinguish between bisimilar graphs that ALC cannot*