1: Conceptual Graphs & Operations

CG Discussion 01 – March 26, 2010

Aziza Mamadolimova

History in two lines

CGs have been developed as a graphic representation for logic with the full expressive power of first-order logic and with extensions to support meta-language, modules, and namespaces.

Conceptual graphs are a formalism for knowledge representation.

Conceptual Graphs were introduced by John F. Sowa in 1976.

The first book on Conceptual Graphs **[Sowa]** applied them to a wide range of topics in artificial intelligence, computer science, and cognitive science.

[Sowa] Sowa, John F. *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA, 1984.

Knowledge representation & reasoning formalism

- The problem is to symbolically encode human knowledge and reasoning in such a way that this encoded knowledge can be processed by a computer via encoded reasoning to obtain intelligent behavior.

- Knowledge can be symbolically represented in many ways. One of them is a graph formalism – knowledge is represented by *labeled graphs*, in graph theory sense, and reasoning mechanisms are based on *graph operations*.

Logically founded

- One of the requirements for a Knowledge Representation Formalism is *to be logically founded*, i.e., each formula in some Formal Logic is represented by a Knowledge Representation Formalism.

Basic Definitions

- The basic structure for representing background knowledge for human-like inference is called the *schema*.
  Schema is a pattern derived from past experience that is used for interpreting, planning, and imagining other experiences. In other words, it's a structure that organize our knowledge and assumptions about something and is used for interpreting and processing information.

- A *prototype* is a typical instance.

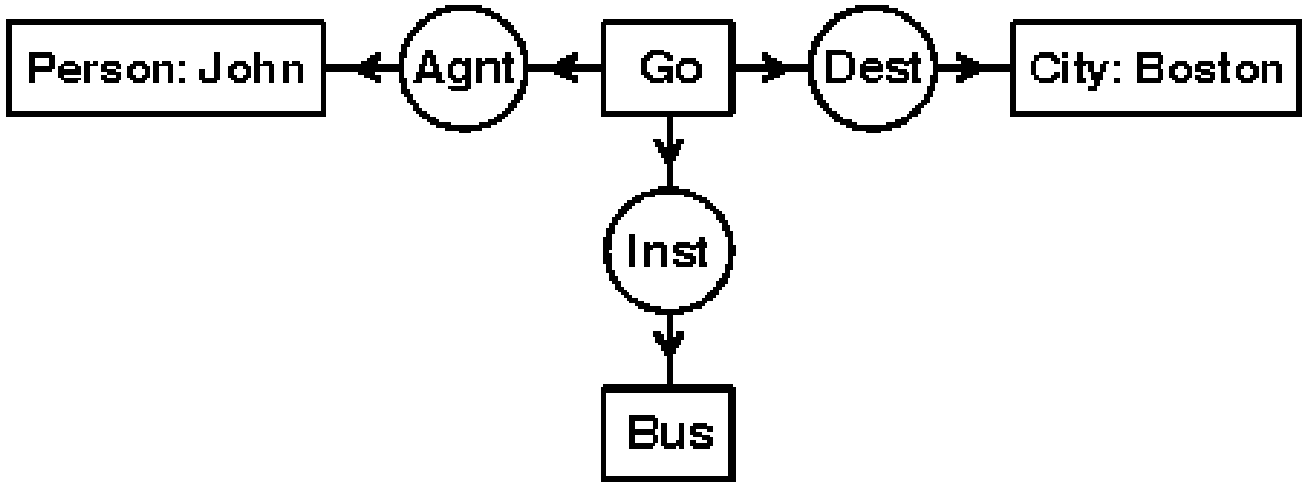
Basic Definitions

- A **type** is a label or name we give to group of entities with similar traits. If we can categorize a number of individuals (e.g., “John”, “Alfred”, ”Mary”) in the same group(e.g., “Person”), then we can call the name of the group, together with the definition of the group, a “type”.

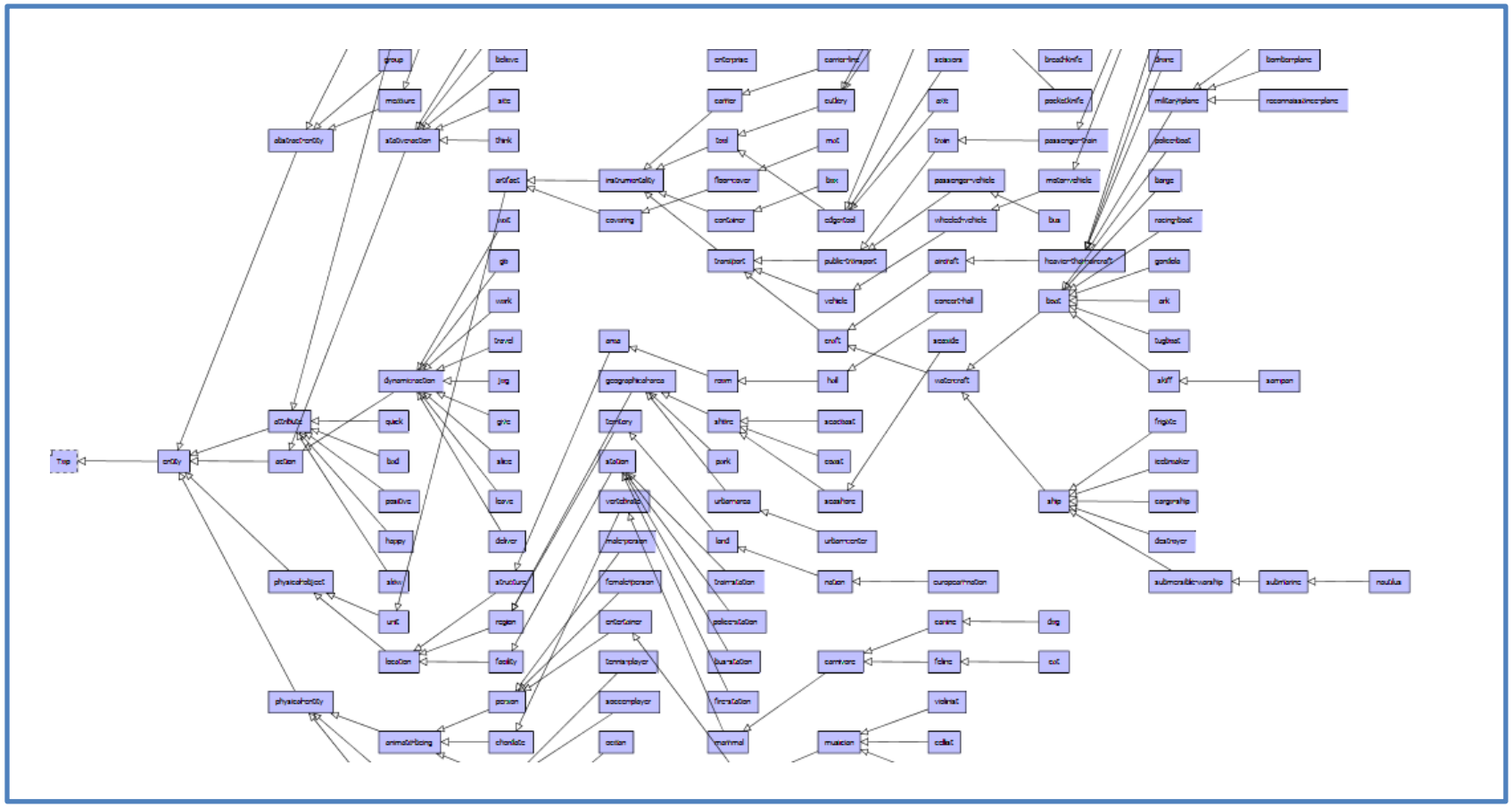
- A **relation** type is simply a name which we give to relation . It tells us what kind of relation we are dealing with. The relation type also determines the valence(number of arcs that belong to it) of the relation and its signature.

CG Definition

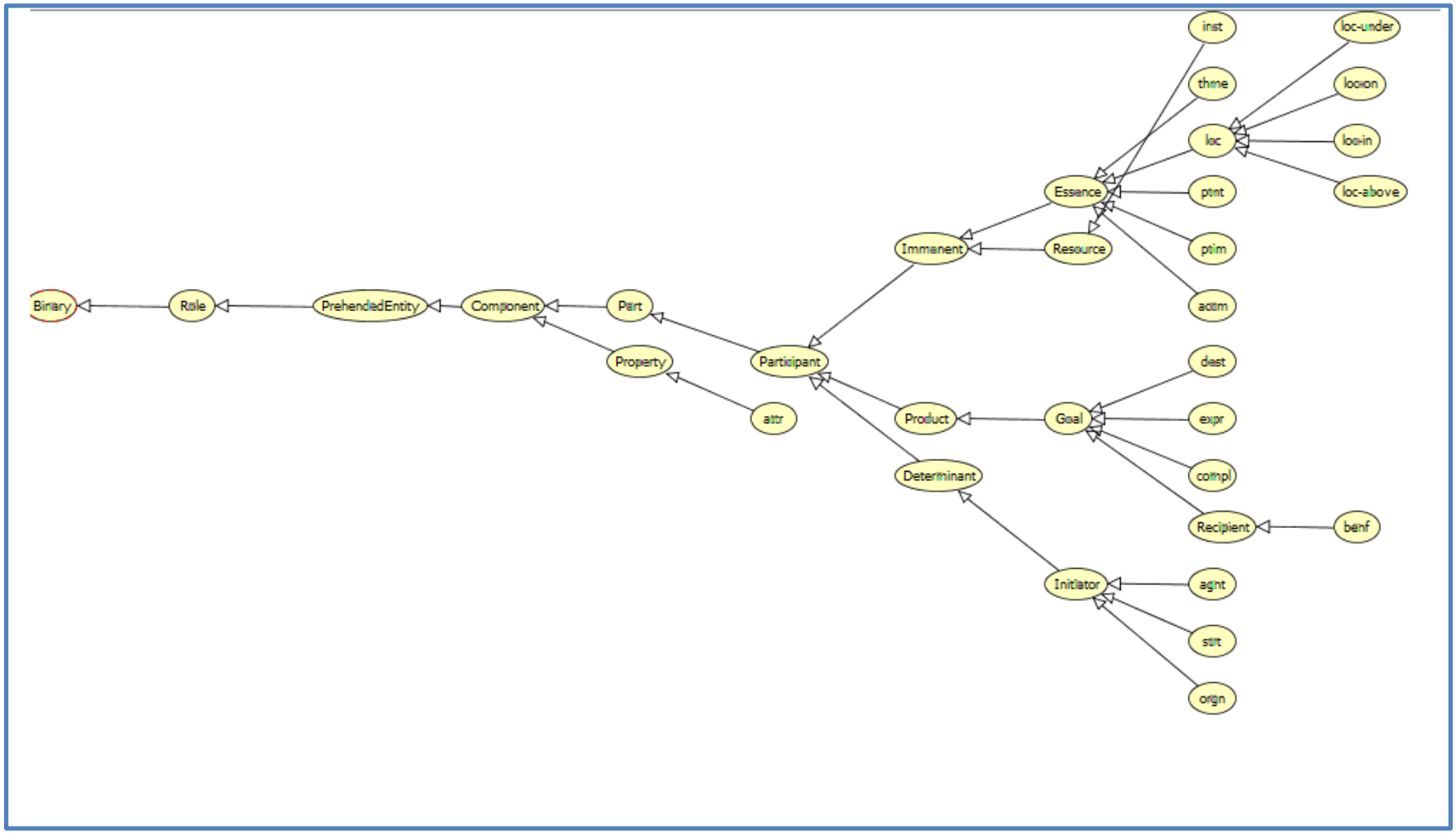
A **conceptual graph** g is a bipartite graph, which consists of two kinds of nodes called **concepts** and **conceptual relations**.



Concept Type Hierarchy



Relation Type Hierarchy



Examples

[LivingFish: ∀]->(Attr)->[Wet]
"All living fish are wet"

[Person] ->(Has)->[Leg: @2]
"There exist a person who has two legs"

(Past)->[Situation:[Present:{*}]<-(Theme)<-[Give]->(Benf)->[Person:Alfred]
"Presents were given to Alfred"

[Salesman:#]<-(Agnt)<-[Travel]->(Path)->[City: {Frederikshavn, Aalborg, Aars}]
"The salesman travels via Frederikshavn, Aalborg, Aars"

Quantifiers

∀ - means “for all”

{*} - means set of things, can be extended {*}@40

@ - to represent quantity

Collections, such as “{Romeo, Juliet }”

“#” - means “the” (referring to a specific thing)

Quantifiers

For the linear graph notation, concept box is normally separated by a colon with the ***type field*** on the left and the ***referent field*** on the right. The referent field can be a name, for e.g. **[cat: yojo]**

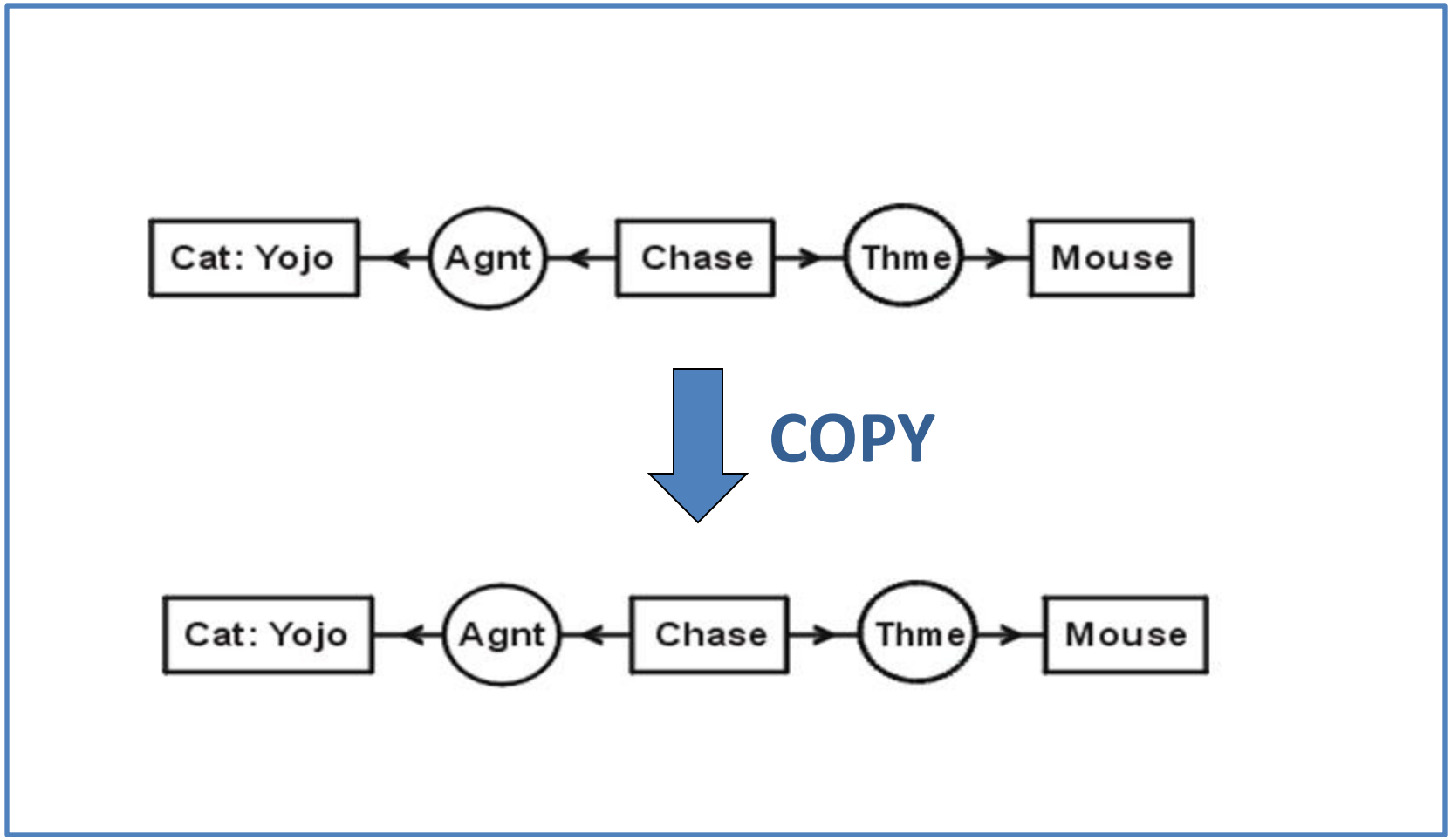
Kind of referent	**Example**	**How it is read**
Universal	**[cat: V]**	Every cat
Singular	**[cat: @1]**	Exactly one cat
Generic Set	**[cat: {*}]**	Cats
Counted Set	**[cat: {*}@3]**	Three cats
Set of individuals	**[cat: {yojo, bell}]**	Yojo and Bell
List of individuals	**[cat: <yojo, bell>]**	Yojo and then Bell
Partial set of individuals	**[cat: {yojo, bell, *}]**	Yojo, Bell and others
Question	**[cat: ?]**	Which cat?
Plural question	**[cat: {*}?]**	Which cats?
Measure	**[interval: @5sec]**	Interval of 5 seconds

Canonical Formation Rules

Not all conceptual graphs make sense. For example:
[SLEEP]->(AGNT)->[IDEA]-(COLR)->[GREEN]
To distinguish the meaningful graphs that represent real or possible situations in the external world, certain graphs are declared to be **canonical**. New canonical graph may be derived from other canonical graphs by the rules **copy**, **restrict**, **join** and **simplify**.

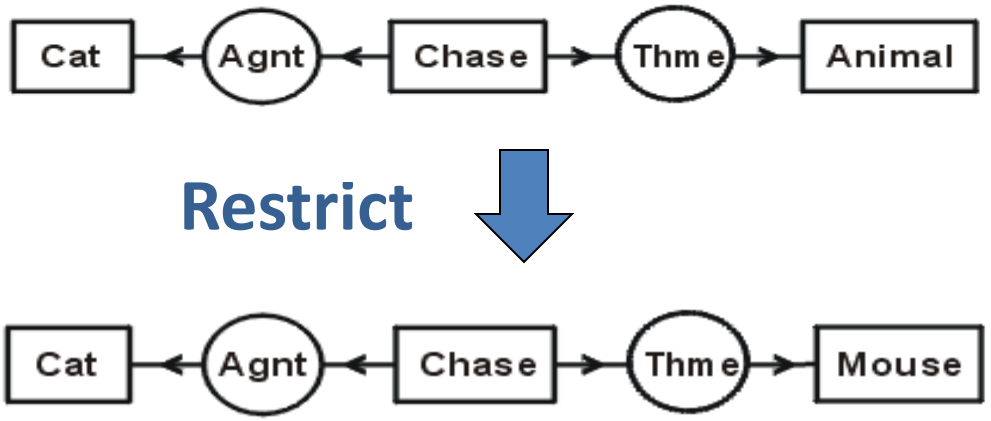
1. **Copy**: Make an exact copy of any CG.
2. **Restrict**: Specialize a type label to a subtype or an existential quantifier to a constant.
3. **Join**: Merge two identical concepts from the same CG or two different CGs.
4. **Simplify**. If conceptual relations **r** and **s** in the graph u are duplicates, then one of them may be deleted from **u** together with all its arcs.

Formation Rule: Copy

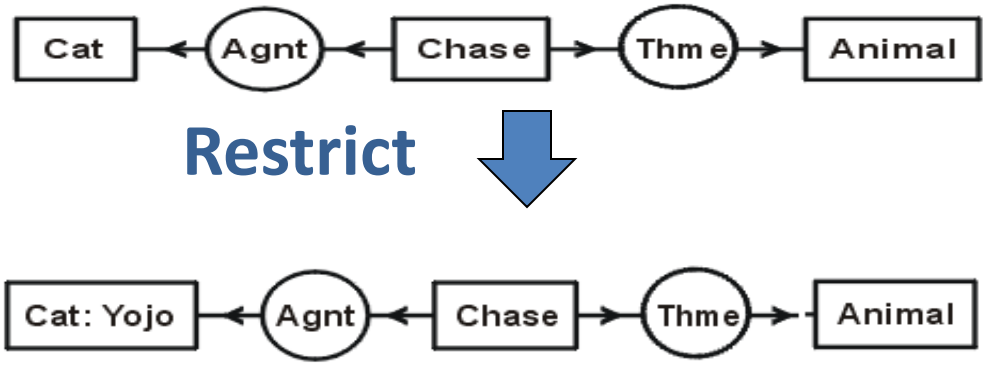


Formation Rule: Restrict

1st case: Specialize a type label to a subtype

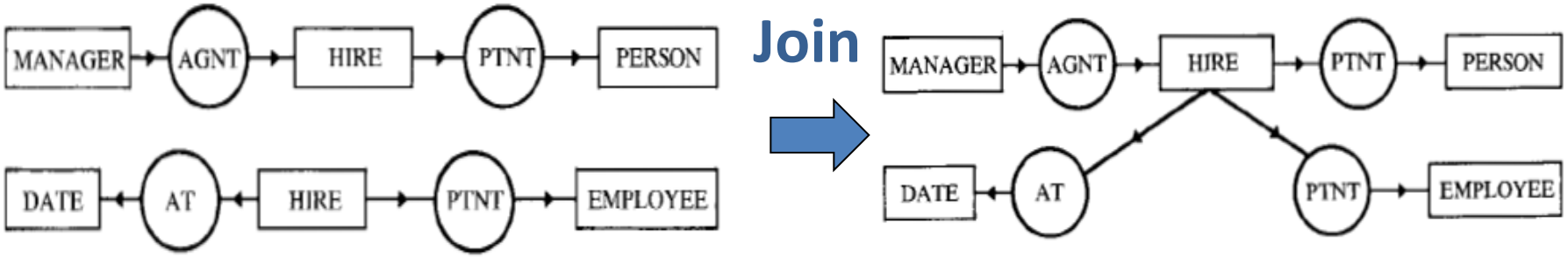


2nd case : Specialize an existential quantifier to a constant



Formation Rule: Join

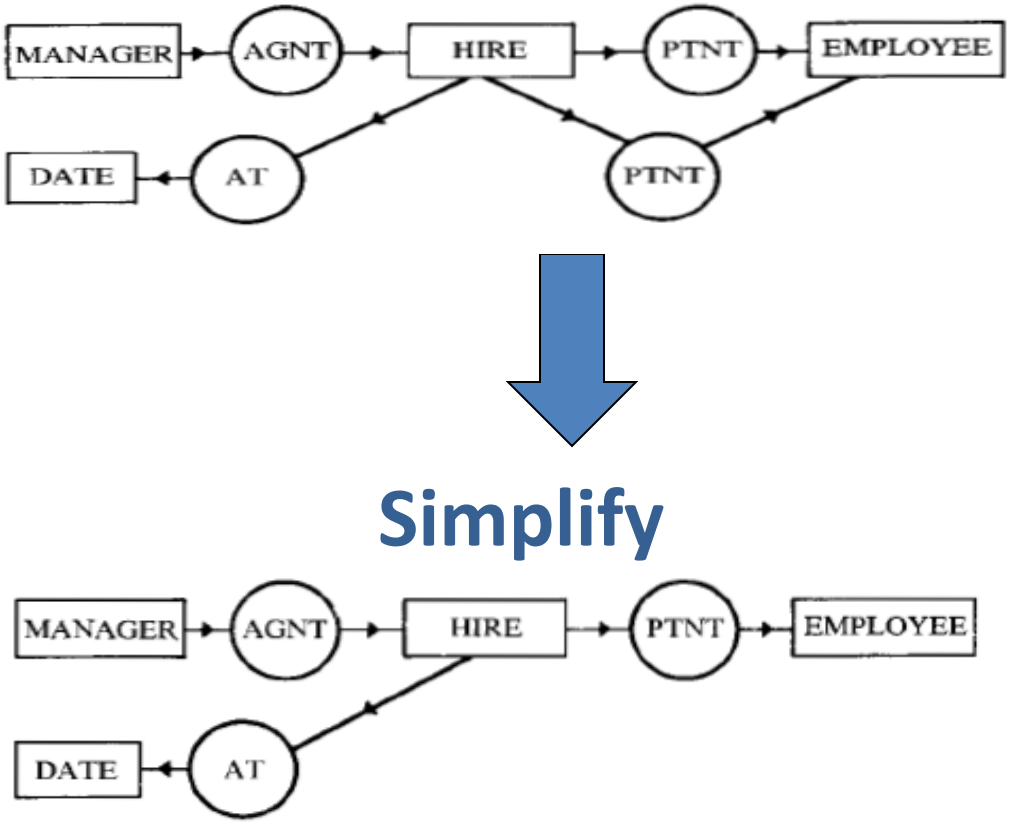
1st case. Merging two identical concepts from two different CGs



2nd case. Merging two identical concepts from the same CG

Formation Rule: Simplify

- One of each pair of duplicates can be deleted by the rule of simplification: when two relations of the same type are linked to the same concepts in the same order, they assert the same information; one of them may therefore be erased.



Additional Formation Rules

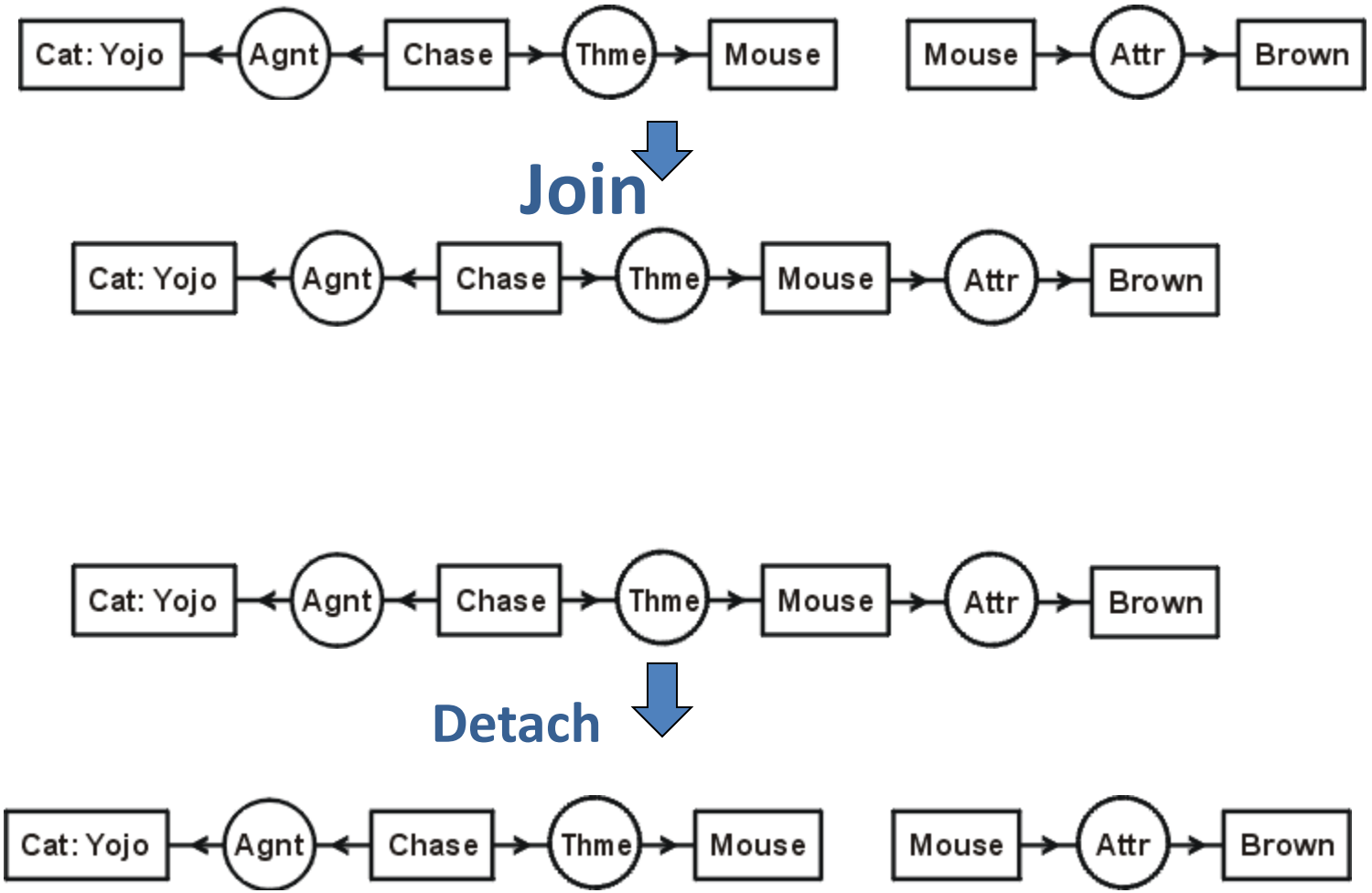
- In 2000 J.Sowa introduced 2 more formation rules: **detach** (which undoes what was done by join rule) and **unrestrict** (makes specialized graph generalized).

- He introduced them as inverse rules to the previously defined ones: detach is inverse to join; unrestrict is inverse to restrict. He also mentioned that these rules are fundamentally graphical: they are easier to show than to describe (Sowa 2008).

Additional Formation Rules

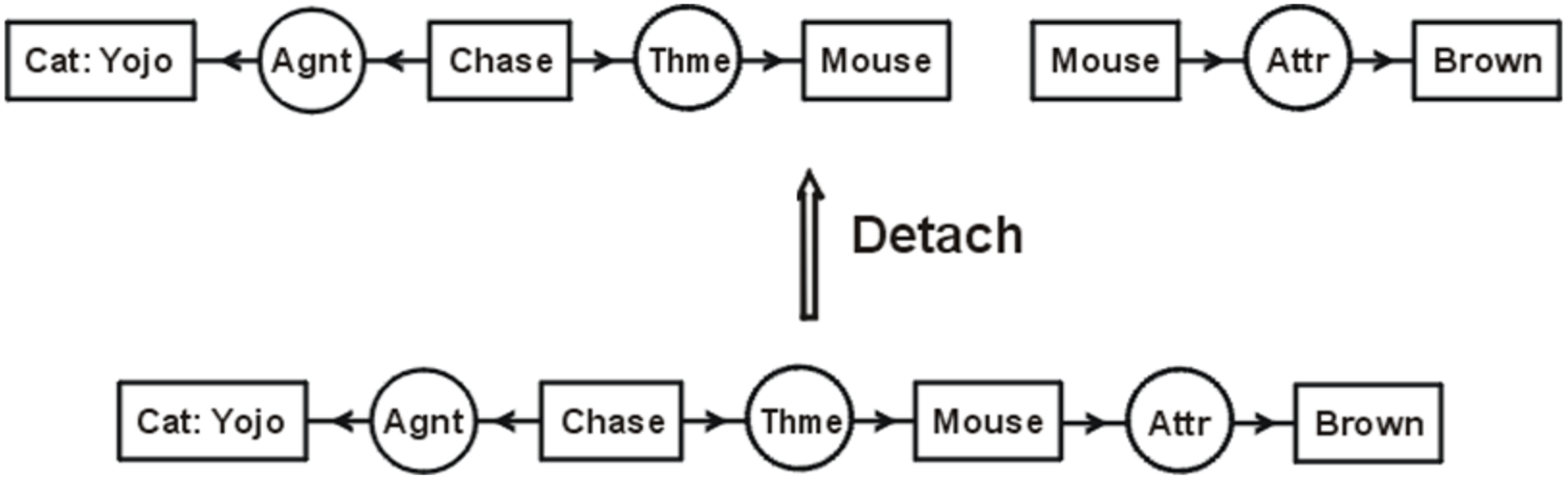
- In 2000 J.Sowa introduced 2 more formation rules: **detach** (which undoes what was done by join rule) and **unrestrict** (makes specialized graph generalized).

- He introduced them as inverse rules to the previously defined ones: detach is inverse to join; unrestrict is inverse to restrict. He also mentioned that these rules are fundamentally graphical: they are easier to show than to describe (Sowa 2008).


Cat: Yojo
Agnt
Chase
Thme
Mouse
Mouse
Attr
Brown
Join
Cat: Yojo
Agnt
Chase
Thme
Mouse
Attr
Brown
Cat: Yojo
Agnt
Chase
Thme
Mouse
Attr
Brown
Detach
Cat: Yojo
Agnt
Chase
Thme
Mouse
Mouse
Attr
Brown

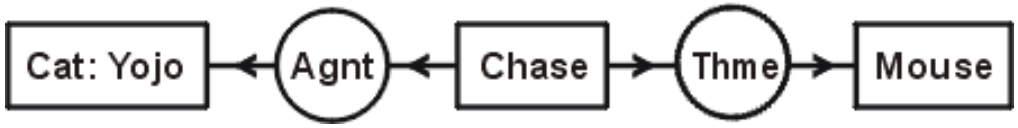
Detach Rules

- In Sowa's 1984 book **detach** rule was defined as following: -it erases any conceptual relation (and all the arcs that belong to it).
- But in 2008 book he used **detach** for following example for splitting the graph into two pieces.

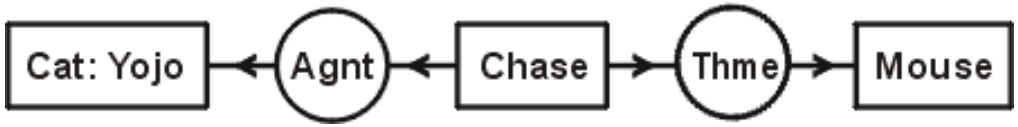


- According to his explanation “a new definition subsumes the old one”, which means that :
- Starting from the bottom graph, erase the (Attr) relation.(That produces the graph on the left at the top, plus another graph that consists of just one concept [Brown]).
- Start from the same bottom graph and erase the relation (Thme).
- That produces the graph on the right at the top, plus another graph for the sentence "Yojo chases".

Restrict



Unrestrict



Specialization and Generalization

Join and restriction are specification rules.

Detach and unrestrict are generalization riles.

Definition: If a conceptual graph **u** is canonically derivable by specialization rules from a conceptual graph **v** , then **u** is called a specialization of **v**, written **u ≤ v**, and **v** is called a generalization of **u**.

[ENTITY] <- (AGNT) <-[EAT] **v**

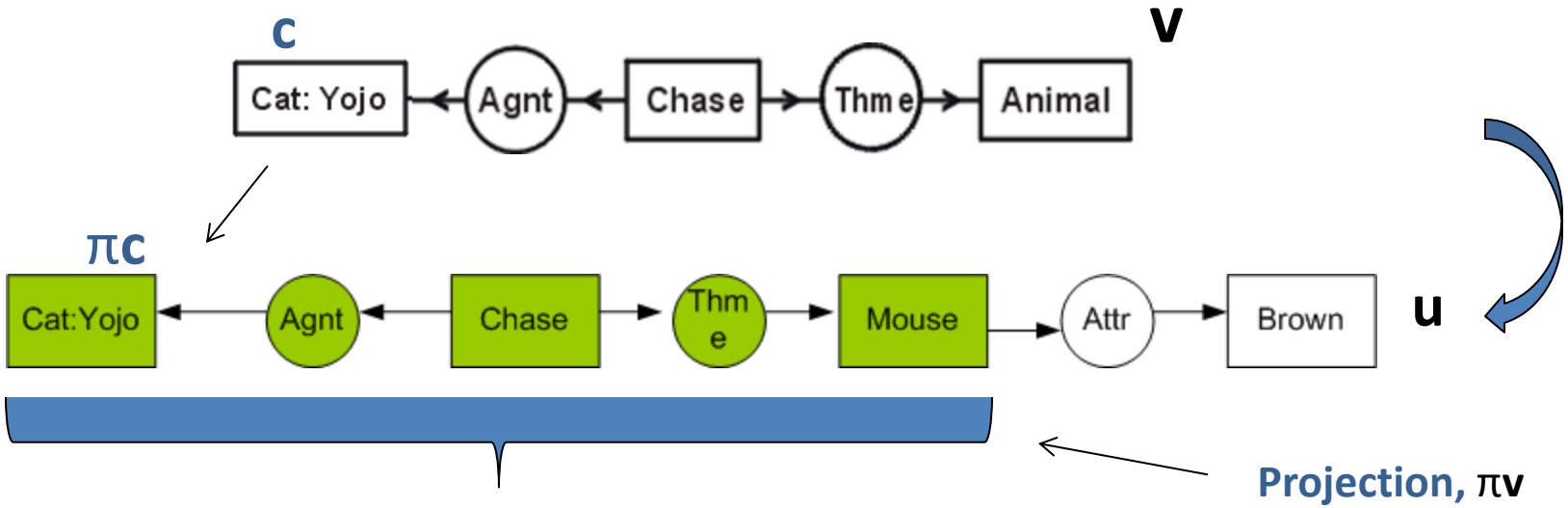
[PERSON: Sue] <- (AGNT) <- [EAT] -> (OBJ) -> [PIE] **u**

[GIRL: Sue] <- (AGNT) <- [EAT] - -> (MANR) ->[FAST]-> (OBJ) -> [PIE] **u1**

Projection

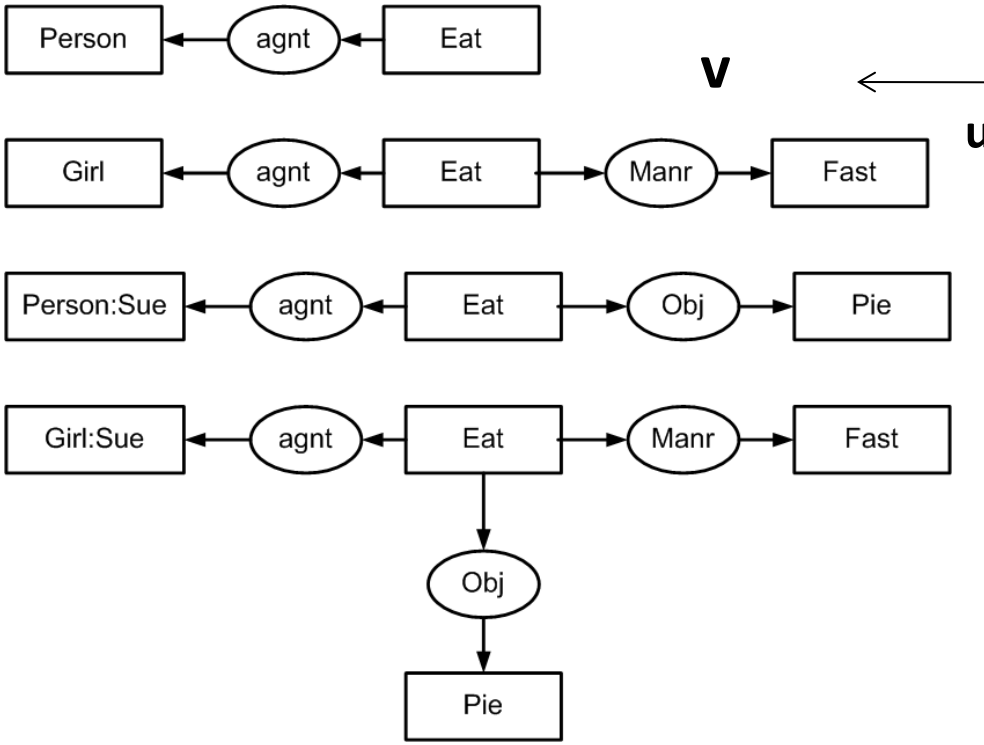
Definition: For any conceptual graphs **u** and **v** where **u** ≤ **v**, there must exist a mapping π **: v -> u**, where π**v** is a subgraph of **u** called a projection of **v** in **u**. The projection operator π has the following properties:

- For each concept **c** in **v**, π**c** is a concept in π**v** where **type(**π**c) ≤ type (c)**. If **c** is individual, then **referent(c) = referent(**π**c)**.
- For each conceptual relation in **v,** π**r** is a conceptual relation in π**v** where **type(**π**r) = type (r)**. If the **i**th arc of **r** is linked to a concept **c** in **v**, the **i**th arc of π**r** must be linked to π**c** in π**v**.



Common generalization & specialization

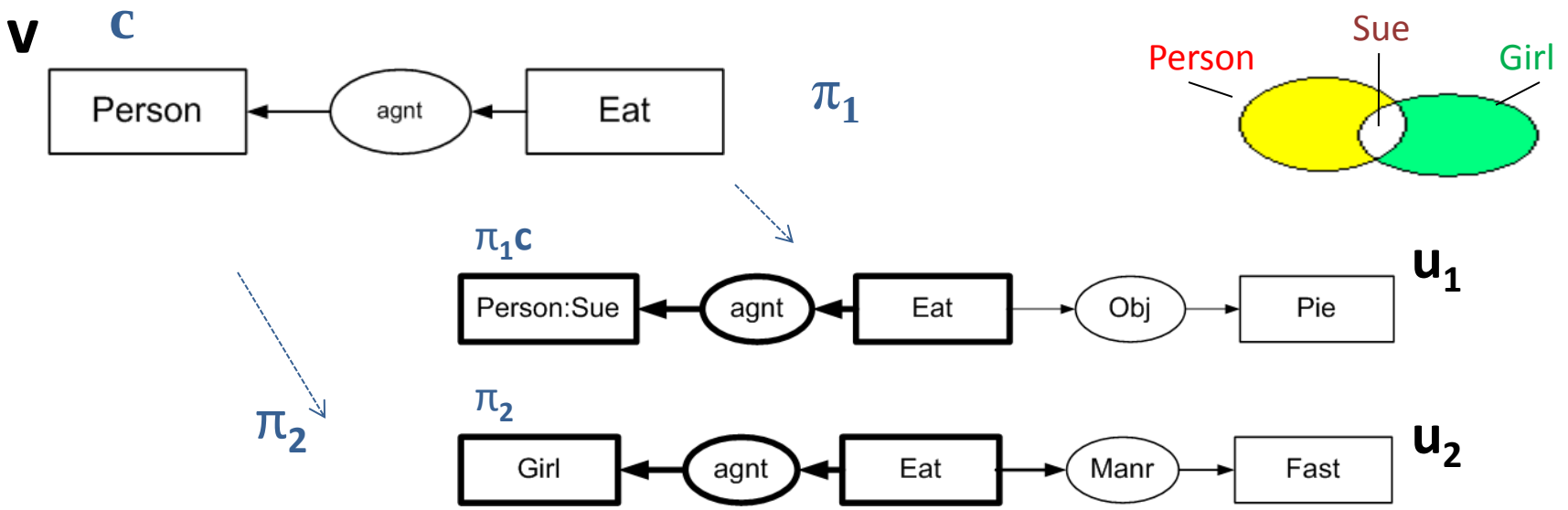
Consider the following four graphs:



- **Definition**. Let $\mathbf{u}_1$, $\mathbf{u}_2$,$\mathbf{v}$ and $\mathbf{w}$ be conceptual graphs. If $\mathbf{u}_1 \leq \mathbf{v}$ and $\mathbf{u}_2 \leq \mathbf{v}$, then $\mathbf{v}$ is called a common generalization of $\mathbf{u}_1$ and $\mathbf{u}_2$. If $\mathbf{w} \leq \mathbf{u}_1$ and $\mathbf{w} \leq \mathbf{u}_2$, then $\mathbf{w}$ is called a common specialization of $\mathbf{u}_1$ and $\mathbf{u}_2$.

Compatible projections

Definition : If two graphs $\mathbf{u}_1$ and $\mathbf{u}_2$ have a common generalization **v**, then the corresponding projections π_1 **v** in $\mathbf{u}_1$ and π_2 **v** in $\mathbf{u}_2$ are candidates for being merged by a series of joins. Such a merger might be blocked, however, by incompatible type labels or referents. If there are no incompatibilities, then the two projections are said to be **compatible**.



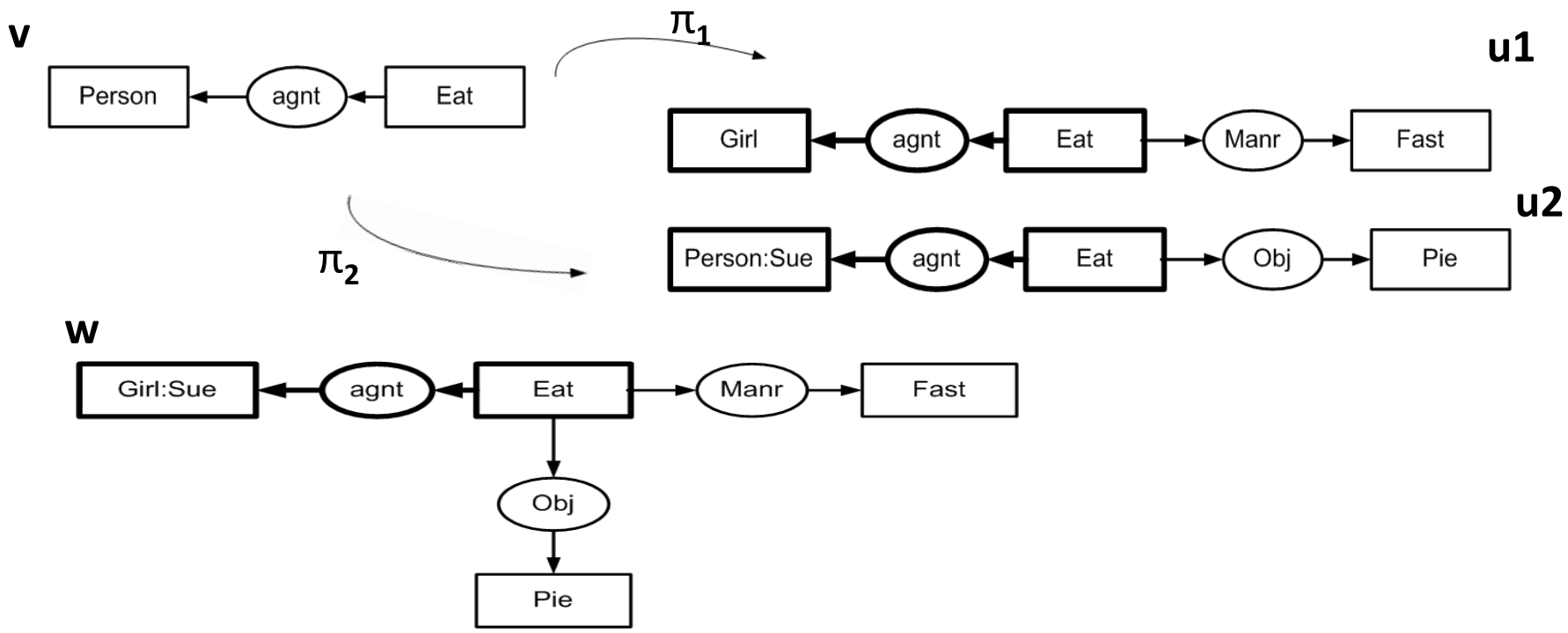
Join on compatible projections

Theorem. If conceptual graphs $\mathbf{u}_1$ and $\mathbf{u}_2$ have a common generalization **v** with compatible projections π_1 **: v ->** $\mathbf{u}_1$ and π_2 **: v ->** $\mathbf{u}_2$, then there exists a unique conceptual graph **w** with the following properties:

w is a common specialization of $\mathbf{u}_1$ and $\mathbf{u}_2$.

There exist projections π_1 **':**$\mathbf{u}_1$ **->** **w** and π_2 **':**$\mathbf{u}_2$ **->w where** π_1 **'** π_1 **v=** π_2 **'** π_2 **v.**

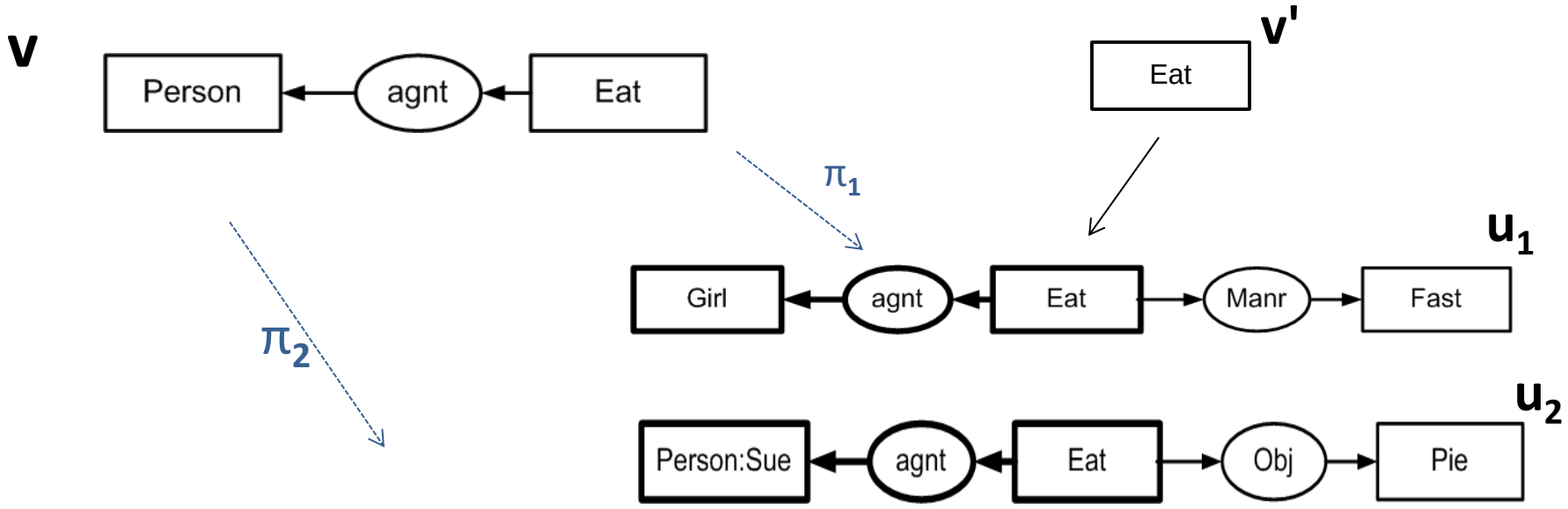
The graph **w** is called a **join** on compatible projections of $\mathbf{u}_1$ and $\mathbf{u}_2$.



Extensions

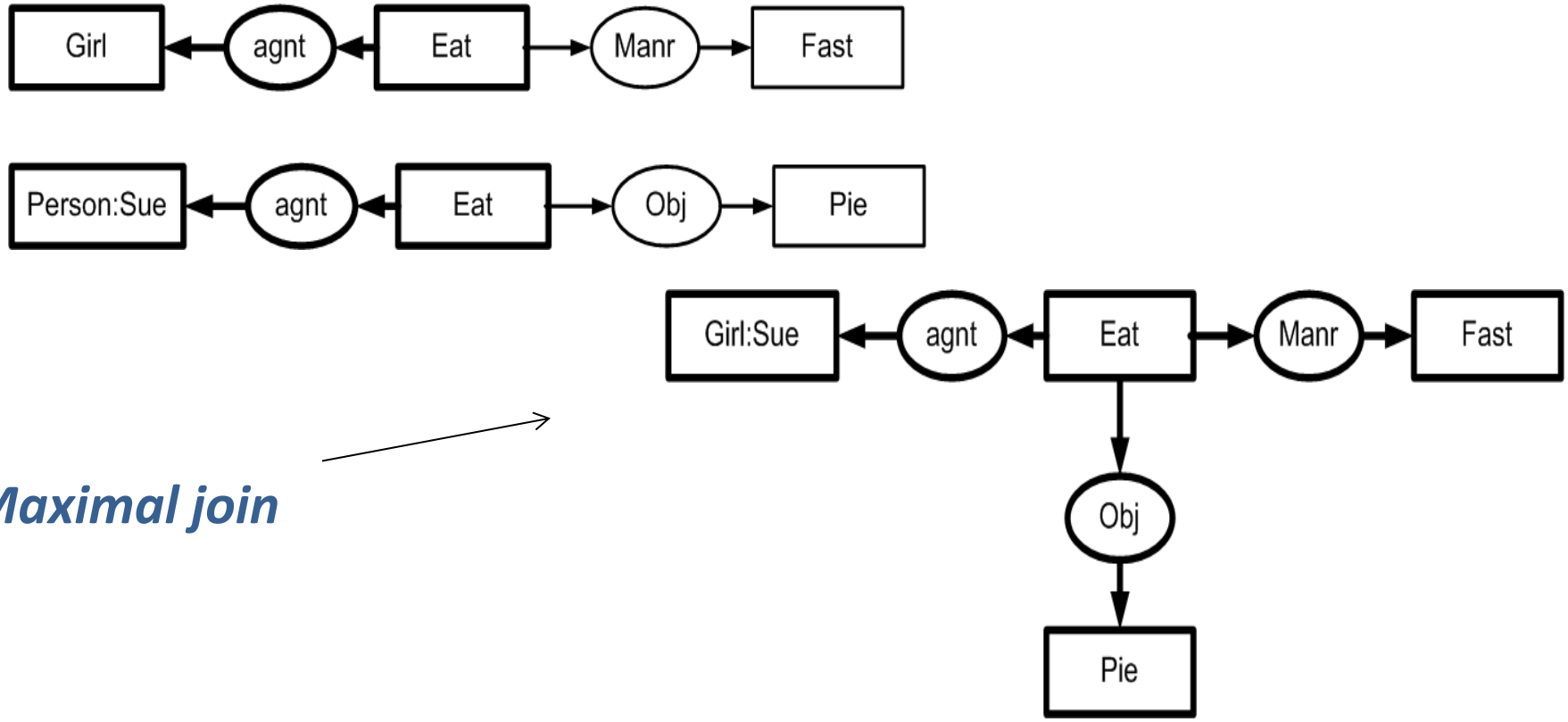
Definition : Let conceptual graphs $\mathbf{u}_1$ and $\mathbf{u}_2$ have a common generalization **v** with compatible projections π_1 **: v->**$\mathbf{u}_1$ and

π_2 **: v->**$\mathbf{u}_2$, and let **v'** be a subgraph of **v.** Then **v'** is also a common generalization of $\mathbf{u}_1$ and $\mathbf{u}_2$ with compatible projections π_1 **: v' -> u**$_1$ and π_2 **: v' ->u**$_2$. The compatible projections π_1 **v** and π_2 **v** are said to be **extensions** of π_1 **v'** and π_2 **v'**.



Maximal Join

Definition. Two compatible projections are said to be maximally extended if they have no extensions. A join on maximally extended compatible projections is called a *maximal join*.



Maximal join