Foundations of Knowledge Representation in Cyc

- ***Why use logic?***
- CycL Syntax
- Collections and Individuals (#$isa and #$genls)
- Microtheories

CYCORP

NL vs. Logic: Expressiveness

NL:

Jim's injury resulted from his falling.

Jim's falling caused his injury.

Jim's injury was a consequence of his falling.

Jim's falling occurred before his injury.

NL: Write the rule for every expression?

Logic: identify the common concepts, e.g.

the relation: x caused y

Write rules about the common concepts, e.g.

x caused y → x temporally precedes y

NL vs. Logic: Ambiguity and Precision

NL: Ambiguous

- x is at the bank.
  - river bank?
  - financial institution?

- x is running.
  - changing location?
  - operating?
  - a candidate for office?

Logic: Precise

x is running-InMotion → x is changing location
x is running-DeviceOperating → x is operating
x is running-AsCandidate → x is a candidate

Reasoning: Figuring out what must be true, given what is known. Requires precision of meaning.

CYCORP

NL vs. Logic:Calculus of Meaning

Logic: Well-understood operators enable reasoning:

Logical constants: not, and, or, all, some

Not (All men are taller than all women).

<u>All men are taller than 12”.</u>

Some women are taller than 12”.

Not (All A are F than all B).

<u>All A are F than x.</u>

Some B are F than x.

CYCORP

Logic-Based Language vs. Other Formal Languages

- Frames & Slots, OO
  - Reasoning depends on mode (in-args, out-args)
  - Less reuse; either less coverage or more bulk and more work
  - KR must be designed around indexing
  - Implicit knowledge is code-dependent

- Logic
  - Mode-independent
  - KR and Indexing are independent
  - Implicit knowledge is preserved in the KB

CYCORP

Indexing and KR

carl

animal_type: elephant

mother: claire

(isa Carl Elephant)

(mother Carl Claire)

claire

animal_type: elephant

mother: elaine

elephant

order: mammal

(genls Elephant Mammal)

Implicit knowledge

elephant

order: mammal

*size: large

*color: gray

*height:

*weight:

(genls Elephant Mammal)

(implies

(and

(isa ?X Elephant)

(gender ?X Male)

(height ?X (Meter ?Y)))

(weight ?X (Ton (TimesFn ?Y 2))))

Summary

- Why use logic?
  - Expressiveness
  - Precision
  - Meaning
  - Use-neutral representation
- Indexing and KR
- Implicit knowledge

CYCORP

Foundations of Knowledge Representation in Cyc

- Why use logic?
- ***CycL Syntax***
- Collections and Individuals (#$isa and #$genls)
- Microtheories

Syntax: Constants

CycLConstants denote specific individuals or collections

(relations, people, computer programs, types of cars . . .)

Each CycLConstant is a character string prefixed by **#$**

- A sampling of some constants:
  - **#$Dog, #$SnowSkiing, #$PhysicalAttribute**
  - **#$BillClinton,#$Rover, #$DisneyLand-TouristAttraction**
  - **#$likesAsFriend, #$bordersOn, #$objectHasColor, #$and, #$not, #$implies, #$forAll**
  - **#$RedColor, #$Soil-Sandy**

These denote collections

These denote individuals :

•Partially Tangible Individuals

•Relations

•Attribute Values

CYCORP

Syntax: Formulas

CycLFormula: a relation applied to some arguments, enclosed in parentheses

- Examples:
  - **(#$isa #$GeorgeWBush #$Person)**
  - **(#$likesAsFriend #$GeorgeWBush #$AlGore)**
  - **(#$BirthFn #$JacquelineKennedyOnassis)**

A CycL Sentence is a well-formed CycLFormula with a Truth Function, such as a predicate in the arg0 position. Sentences have truth values.

A CycL Non-atomic Term is a well-formed CycLFormula with a Function-Denotational in the arg0 position.

CYCORP

Syntax: Sentences

A TruthFunction:

– is a relation that can be used to form sentences.
– begins with a lower-case letter.

- Types of TruthFunctions:
  - Predicates: **#$likesAsFriend, #$bordersOn, #$objectHasColor, #$isa**
  - Logical Connectives: **#$and, #$or, #$not**
  - Quantifiers: **#$implies#$forAll, #$thereExists**

- Sample CycLSentences:
  – **(#$isa #$GeorgeWBush #$Person)**
  – **(#$likesAsFriend #$GeorgeWBush #$AlGore)**

CycLSentences are used to form assertions and queries.

CYCORP

Syntax: Non-atomic Terms

- A Function-Denotational
  - is a relation that can be applied to some arguments to pick out something new
  - usually ends in "Fn"

- Examples of Function-Denotational:
  - **#$BirthFn, #$GovernmentFn, #$BorderBetweenFn**

- Sample CycL Non-atomic Terms:
  - **(#$GovernmentFn #$France)**
  - **(#$BorderBetweenFn #$France #$Switzerland)**
  - **(#$BirthFn #$JacquelineKennedyOnassis)**

CycL Non-atomic Terms are denotational terms. They can be used like any other, as in:

- **(#$residenceOfOrganization (#$GovernmentFn #$France) #$CityOfParisFrance)**

Well-formedness: Arity

- Arity constraints are represented in CycL with the predicate **#$arity**:

  - **(#$arity #$performedBy 2)**

  Represents the fact that #$performedBy takes two arguments, e.g.:
  (#$performedBy #$AssassinationOfPresidentLincoln #$JohnWilkesBooth)

  - **(#$arity #$BirthFn 1)**

  Represents the fact that #$BirthFn takes one arguments, e.g.:
  (#$BirthFn #$JacquelineKennedyOnassis)

CYCORP

Well-Formedness: Argument Type

Argument type constraints are represented in CycL with the following 2 predicates:

1 **#$argIsa**

(#$argIsa #$performedBy 1 #$Action) means that the first argument of #$performedBy must be an individual #$Action, such as the assassination of Lincoln in:

(#$performedBy #$AssassinationOfPresidentLincoln #$JohnWilkesBooth)

2 **#$argGenl**

(#$argGenl #$penaltyForInfraction 2 #$Event) means that the second argument of #$penaltyForInfraction must be a type of #$Event, such as the collection of illegal equipment use events in:

(#$penaltyForInfraction #$SportsEvent #$IllegalEquipmentUse #$Disqualification)

CYCORP

Complex Formulas

- CycL also includes logical terms to allow us to stick formulas together and quantify into them

Logical Terms

Logical Connectives

Quantifiers

Logical Connectives

- Logical connectives
  - truth functions
  - take sentences as their arguments

- **(#$and**
  (#$performedBy #$GettysburgAddress #$Lincoln)
  (#$objectHasColor #$Rover #$TanColor))
- **(#$or**
  (#$objectHasColor #$Rover #$TanColor)
  (#$objectHasColor #$Rover #$BlackColor))
- **(#$implies**
  (#mainColorOfObject #$Rover #$TanColor)
  (#$not (#$mainColorOfObject #$Rover #$RedColor)))
- **(#$not**
  (#$performedBy #$GettysburgAddress #$BillClinton))

CYCORP

Variables and Quantifiers (1)

- By adding variables and quantifiers to the logical connectives, predicates and other CycL components we've already covered, we gain the ability to represent many pieces of ordinary knowledge.
- Sentences involving concepts like "everybody," "something," and "nothing" require variables and quantifiers:

Everybody loves somebody.

Nobody likes spinach.

Some people like spinach and some people like broccoli, but no one likes them both.

CYCORP

Quantifiers

- Adding variables and quantifiers, we can represent more general knowledge.
- Two main quantifiers:

1. **Universal Quantifer** -- **#$forAll**

   Used to represent very general facts, like:

   All dogs are mammals

   Everyone loves dogs

2. **Existential Quantifier** -- **#$thereExists**

   Used to assert that something exists, to state facts like:

   Someone is bored

   Some people like dogs

CYCORP

Quantifiers

- Universal Quantifier

  (#$forAll ?THING
  (#$isa ?THING #$Thing))

  Everything is a thing.

- Existential Quantifier:

  (#$thereExists ?JOE
  (#$isa ?JOE #$Poodle))

  Something is a poodle.

- Others:

  (#$thereExistsExactly 12 ?ZOS
  (#$isa ?ZOS #$ZodiacSign))

  There are exactly
  12 zodiac signs

  (#$thereExistsAtLeast 9 ?
  PLNT (#$isa ?PLNT
  #$Planet))

  There are at least
  9 planets

CYCORP

Implicit Universal Quantification

All variables occurring “free” in a formula are understood by Cyc to be implicitly universally quantified.

So, to CYC, the following two formulas represent the same fact:

```
(#$forAll ?X
  (#$implies
    (#$isa ?X #$Dog)
    (#$isa ?X #$Animal))

(#$implies
  (#$isa ?X #$Dog)
  (#$isa ?X #$Animal))
```

CYCORP

Pop Quiz #1

- What does this formula mean?

```
(#$thereExists ?PLANET
      (#$and
        (#$isa ?PLANET #$Planet)
        (#$orbits ?PLANET #$Sun)))
```

Pop Quiz #1

- What does this formula mean?

```
(#$thereExists ?PLANET
    (#$and
      (#$isa ?PLANET #$Planet)
      (#$orbits ?PLANET #$Sun)))
```

"There is at least one planet orbiting the Sun."

CYCORP

Pop Quiz #2

- What does this formula mean?

```
(#$forAll ?PERSON1
   (#$implies
    (#$isa ?PERSON1 #$Person)
    (#$thereExists ?PERSON2
      (#$and
         (#$isa ?PERSON2 #$Person)
         (#$loves ?PERSON1 ?PERSON2)))
```

CYCORP

Pop Quiz #2

- What does this formula mean?

```
(#$forAll ?PERSON1
   (#$implies
    (#$isa ?PERSON1 #$Person)
    (#$thereExists ?PERSON2
      (#$and
          (#$isa ?PERSON2 #$Person)
          (#$loves ?PERSON1 ?PERSON2)))
```

“Everybody loves somebody.”

CYCORP

Pop Quiz #3

- How about this one?

```
(#$implies
    (#$isa ?PERSON1 #$Person)
    (thereExists ?PERSON2
      (#$and
       (#$isa ?PERSON2 #$Person)
       (#$loves ?PERSON2 ?PERSON1))))
```

Pop Quiz #3

- How about this one?

```
(#$implies
    (#$isa ?PERSON1 #$Person)
    (thereExists ?PERSON2
      (#$and
       (#$isa ?PERSON2 #$Person)
       (#$loves ?PERSON2 ?PERSON1))))
```

“Everyone is loved by someone.”

Pop Quiz #4

And this?

```
(#$implies
    (#$isa ?PRSN #$Person)
    (#$loves ?PRSN ?PRSN))
```

Pop Quiz #4

And this?

```
(#$implies
    (#$isa ?PRSN #$Person)
    (#$loves ?PRSN ?PRSN))
```

“Everyone loves his (or her) self.”

Denotational Functions

- Denotational Functions can be applied to some arguments to pick out something new. The result of applying a denotational function is a term that denotes something. Function names are always capitalized, and often end in “Fn”.

- Examples:
  #$FruitFn
  #$GovernmentFn
  #$DeadFn

Non-atomic Terms

- A non-atomic term (NAT) is a denoting term like a constant.
- NATs are formed by applying a denotational function to a denoting term.

```
(#$FruitFn #$AppleTree)
(#$GovernmentFn #$France)
(#$DeadFn #$Cockroach)
```

- NATs can be used just like atomic terms (i.e., constants).

```
(#$implies
    (#$isa ?APPLE (#$FruitFn #$AppleTree))
    (#$colorOfObject ?APPLE #$RedColor))
```

- The denotation of a NAT is determined by the denotations of the inputs to the denoting function.

CYCORP

Why Use NATs?

- Uniformity
  - All kinds of fruits, nuts, etc., are represented in the same, compositional way:

    (#$FruitFn PLANT) *

- Inferential Efficiency
  - Forward rules can automatically conclude many useful assertions about NATs as soon as they are created, based on the function and arguments used to create the NAT.
    - what kind of thing that NAT represents
    - how to refer to the NAT in English
    - …

CYCORP

Reifiable Functions and NARTS

- Some functions return concepts that we want to "reify" and keep in the KB. These are reifiable functions, such as:

  #$GovernmentFn

  #$BirthFn

- When a new NAT is created using a reifiable function, that new term is itself reified (kept around separately in the KB) and becomes a Non-Atomic Reified Term, or NART, such as

  (#$GovernmentFn #$France)

  (#$BirthFn #$JacquelineKennedyOnassis)

- Other functions return concepts that we *don't* want to store separately in the KB. These are unrefiable functions, such as:

  #$TimesFn

- When a new NAT is created using an unrefiable functions, it does not get created as a persistent term in the KB. Unrefiable functions result in Non-Atomic Unreified Terms, such as **(#$TimesFn 3 7)**.

Summary

- CycL components
  - Constants
  - Formulas
  - Sentences (and Truth Functions)
  - Non-atomic Terms (and Denotational Functions)
  - Logical Constants
  - Variables and Quantifiers
- Well-Formedness
  - arity
  - argument constraints

CYCORP

Foundations of Knowledge Representation in Cyc

- Why use logic?
- CycL Syntax
- ***Collections and Individuals*** ***(#$isa and #$genls)***
- Microtheories

Collections and Individuals

- A ***collection*** is a kind or class.
- Collections have ***instances***.
- Each collection is characterized by some feature(s) that all of its instances share.
- Some collections
  - **#$Tower**
  - **#$SpaceStation**
  - **#$Director-Movie**
  - **#$Person**

CYCORP

Individuals

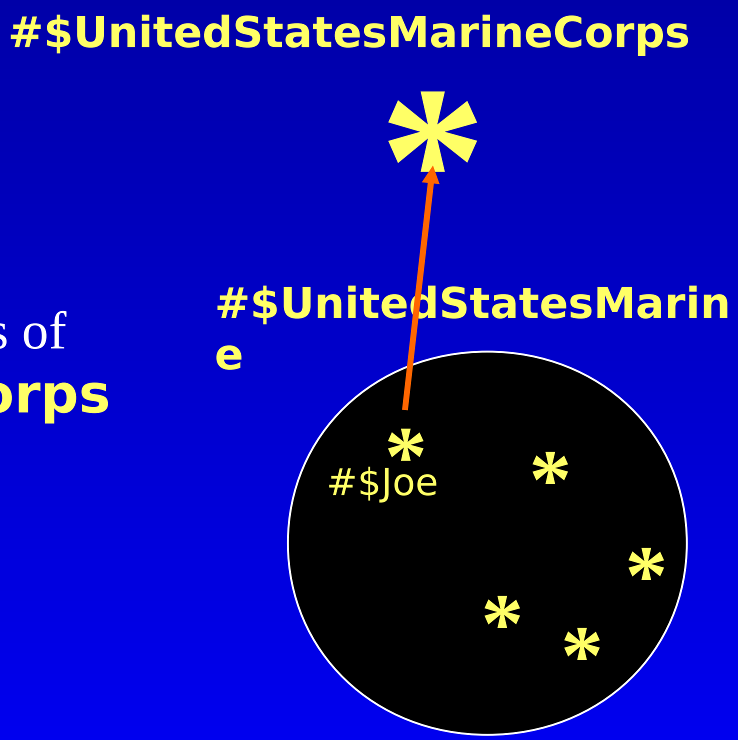
- An individual is a single thing, *not* a collection.
- Individuals do *not* have instances.
- Individuals may have *parts*.
- Some individuals:
  - **#$EiffelTower**
  - **#$Mir**
  - **#$OrsonWelles**
  - **#$UnitedStatesMarineCorps**

*
#$Cher

CYCORP

Joe The Marine

- **#$UnitedStatesMarineCorps**
  - An individual organization
    - A single, specific thing
    - It has parts, but not instances
- **#$UnitedStatesMarine**
  - The collection of all human members of the **#$UnitedStatesMarineCorps**
  - Has instances, each of which is an individual marine



Remember...

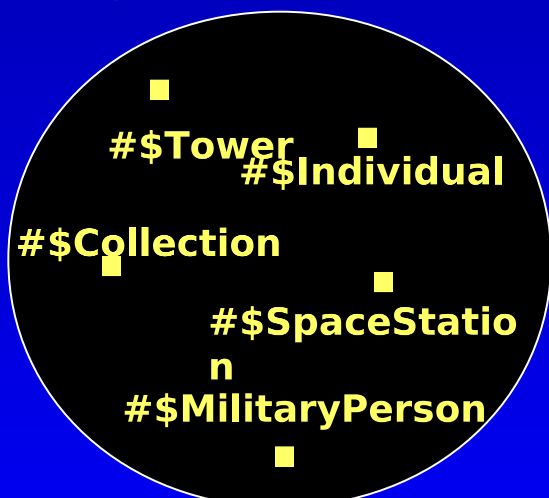
- “Collections can have instances but not parts.”

- “Individuals can have parts but not instances.”

Everything Is An Instance of Something

- Every collection is, at minimum, an instance of **#$Collection**.
- Every individual is, at minimum, an instance of **#$Individual**.

#$Collection



#$Individual

CYCORP

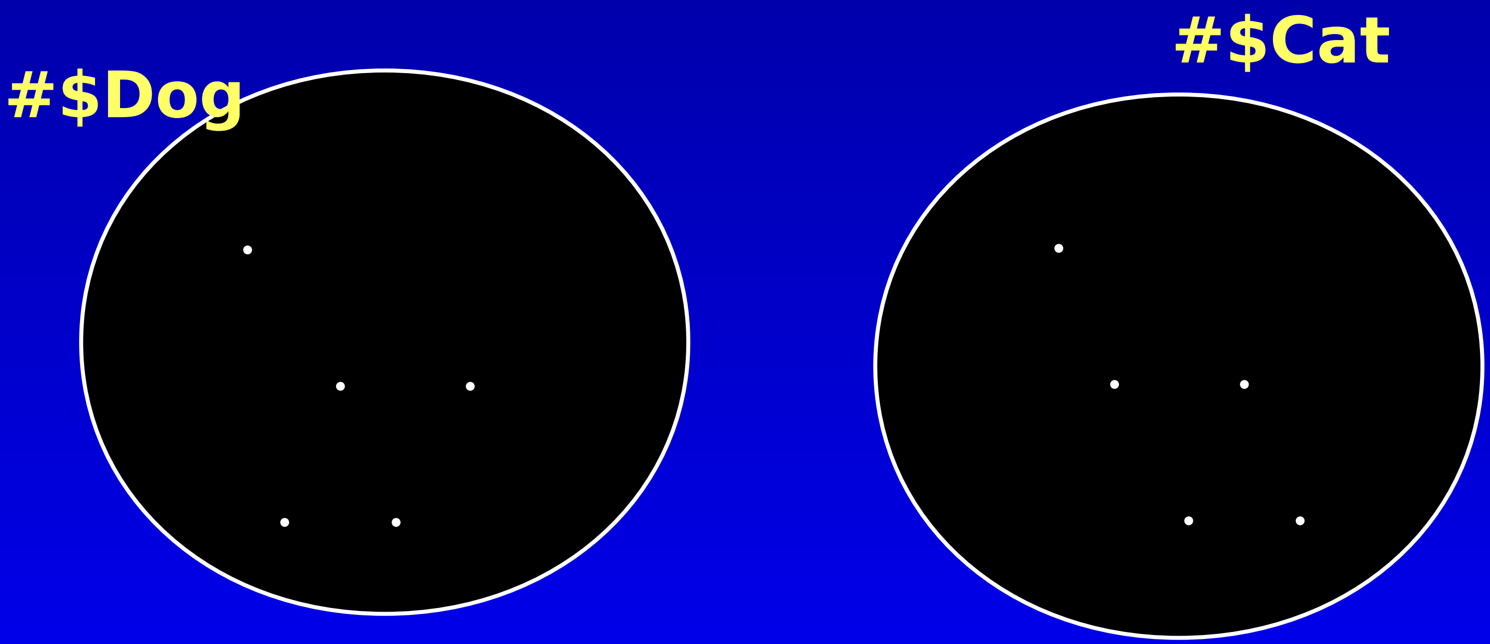
Collections of Collections and Collections of Individuals

- Some collections whose instances are individuals:
  - **#$Tower**
  - **#$Person**
  - **#$Dog**
- Some collections whose instances are collections:
  - **#$ArtifactType**
  - **#$Collection**
- Some collections with instances of both types:
  - **#$ProprietaryConstant**
  - **#$DocumentationConstant**

CYCORP

Disjoint Collections

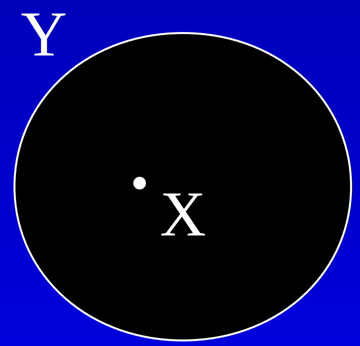
- Collections which have no instances in common are *disjoint*.



(#$disjointWith #$Dog #$Cat)

#$isa

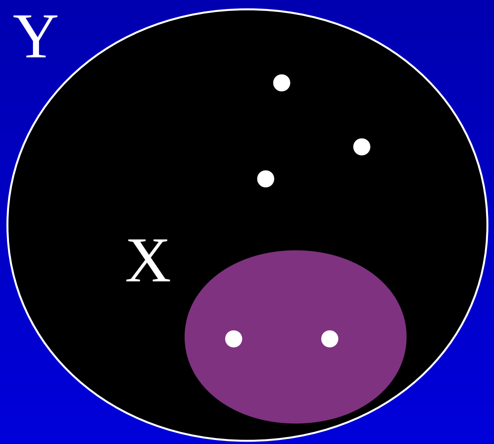
- **(#$isa X Y)** means
  "X is an instance of collection Y."
  - **(#$isa #$EiffelTower #$Tower)**
  - **(#$isa #$Canada #$Country)**
  - **(#$isa #$Cher #$Person)**
  - **(#$isa #$UnitedStatesMarineCorps #$ModernMilitaryOrganization)**



CYCORP

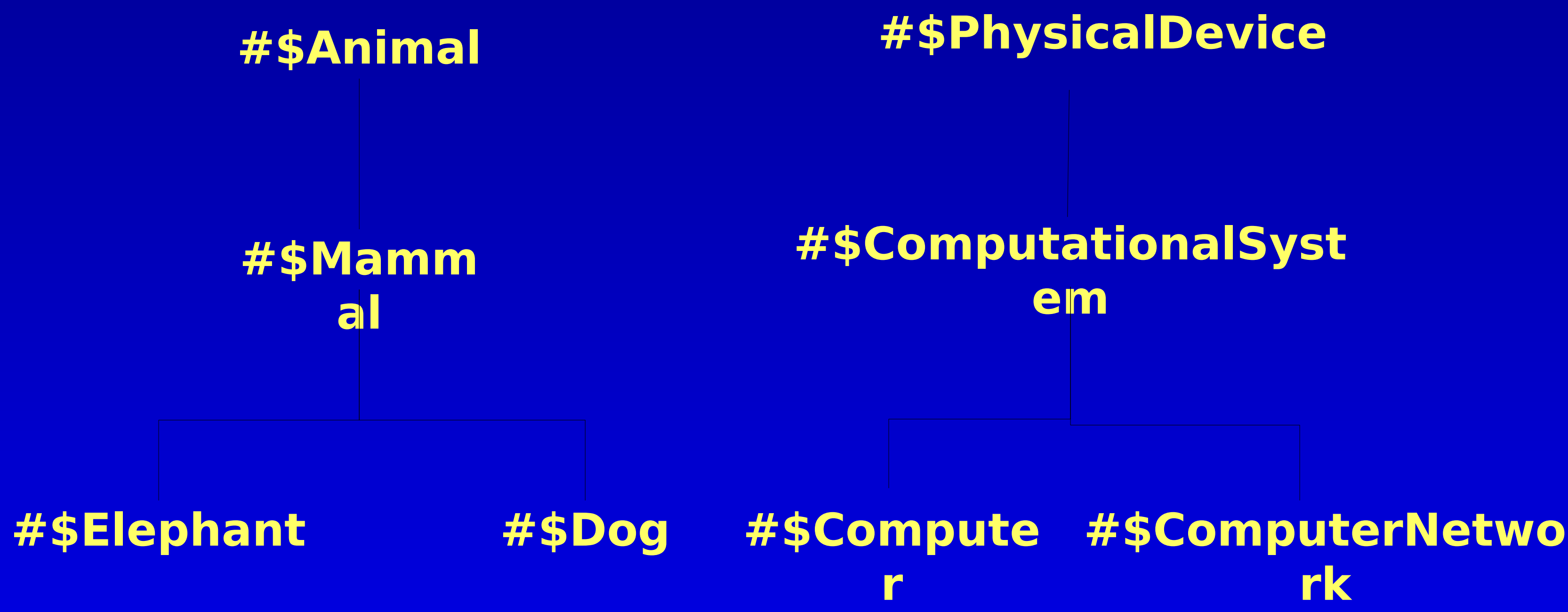
#$genls

- **(#$genls X Y)** means
  "Every instance of collection X is also an instance of collection Y."
  - **(#$genls #$Dog #$Mammal)**
  - **(#$genls #$Tower #$FixedStructure)**
  - **(#$genls #$ModernMilitaryOrganization #$Organization)**

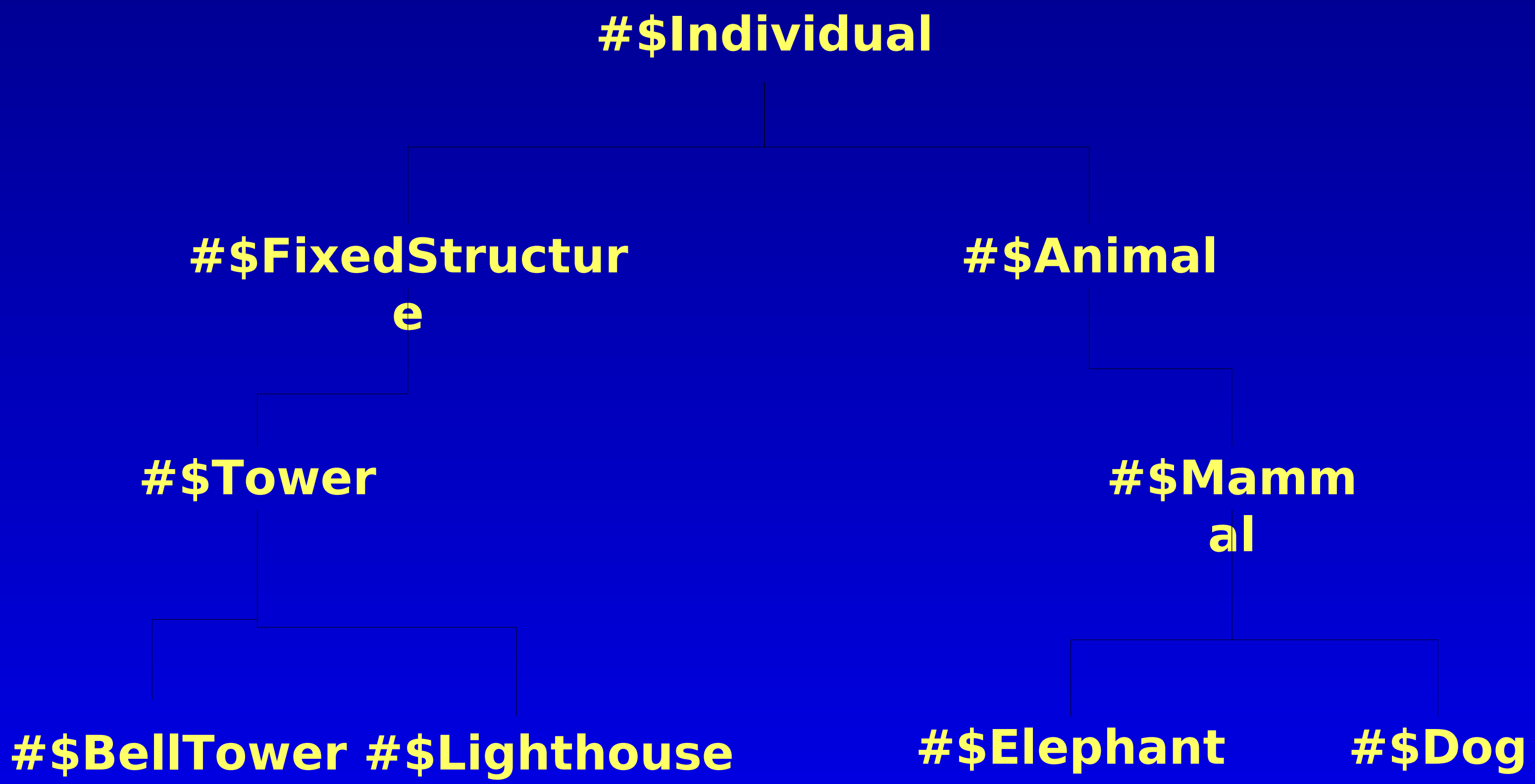


- Sometimes expressed in Cyclish© as:
  "Y is a genls (generalization) of X."
  "X is a spec (specialization) of Y."

#$genls is transitive



The #$genls hierarchy



#$isa is NOT transitive

Remember . . .

- Because every instance of a collection is also an instance of the collection's genls, the following statements are true:
  - "**#$isa** transfers through **#$genls**."
  - "**#$isa** does NOT transfer through **#$isa**."

What can we conclude about #$Rover the dog?

A more complete list of collections of which **#$Rover** is an instance

- **#$Agent-Generic #$Agent #$AirBreathingVertebrate #$Animal #$AnimalBLO #$BilateralObject #$BiologicalLivingObject #$CanineAnimal #$Carnivore #$CompositeTangibleAndIntangibleObject #$Dog #$Eutheria #$Individual #$IndividualAgent #$LeftAndRightSidedObject #$Mammal #$NaturalTangibleStuff #$NonPersonAnimal #$OrganicStuff #$Organism-Whole #$PartiallyIntangible #$PartiallyIntangibleIndividual #$PartiallyTangible #$PerceptualAgent #$SentientAnimal #$SomethingExisting #$SpatialThing #$SpatialThing-Localized #$TemporalThing #$Thing #$Vertebrate**

CYCORP

Is #$genls reflexive?

Consider

(#$genls #$Dog #$Dog)

This means

“Every instance of #$Dog
is an instance of #$Dog.”

Yes!

Is #$isa reflexive?

Consider

(#$isa #$Dog #$Dog)

NOT reflexive

However, consider

(#$isa #$Collection #$Collection)

Not anti-reflexive, either

CYCORP

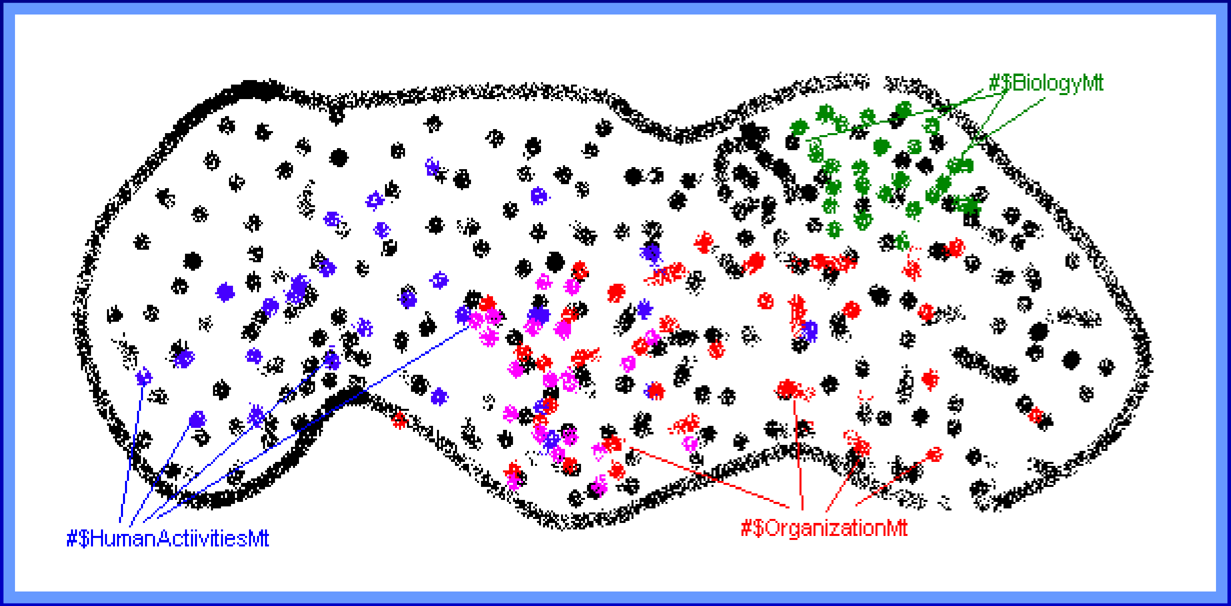
Summary

- Collections vs. Individuals
- #$isa vs. #$genls
- #$genls is transitive
- #$genls is reflexive

Foundations of Knowledge Representation in Cyc

- Why use logic?
- CycL Syntax
- Collections and Individuals (#$isa and #$genls)
- ***Microtheories***

A Bundle of Assertions



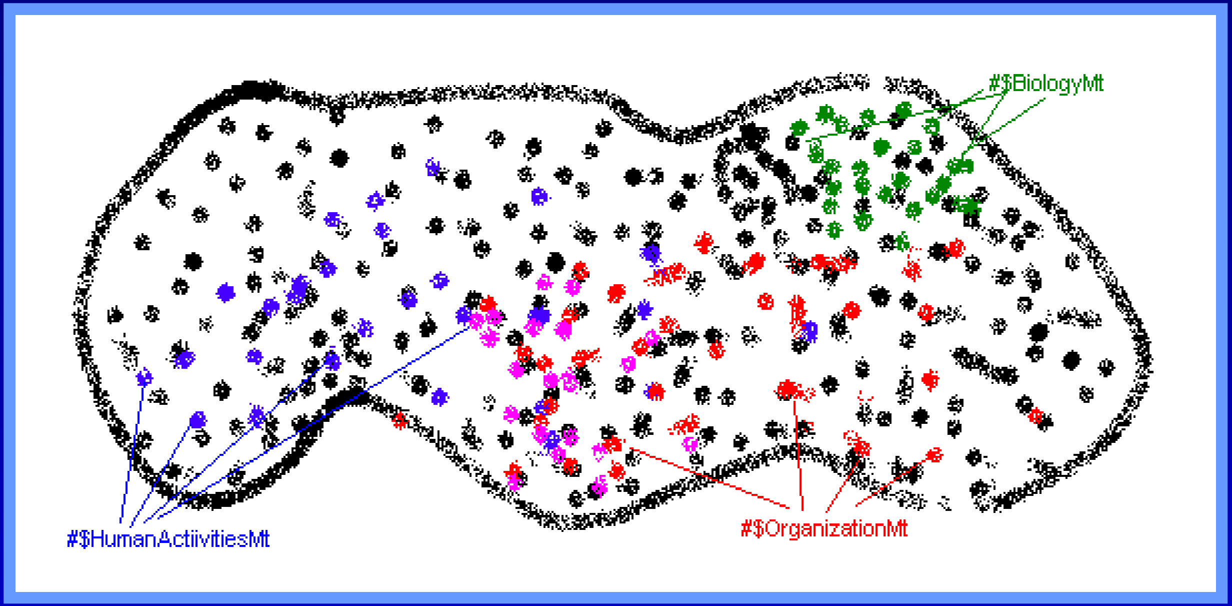
the Cyc KB, as a sea of assertions

- Think of a microtheory (mt) as a set of assertions.

- Each microtheory bundles assertions based on
  - a shared set of assumptions on which the truth of the assertions depends, or
  - a shared topic (world geography, brain tumors, pro football), or
  - a shared source: (CIA World Fact Book 1997, FM101-5, USA Today)

CYCORP

Avoiding Inconsistencies



the Cyc KB, as a sea of assertions

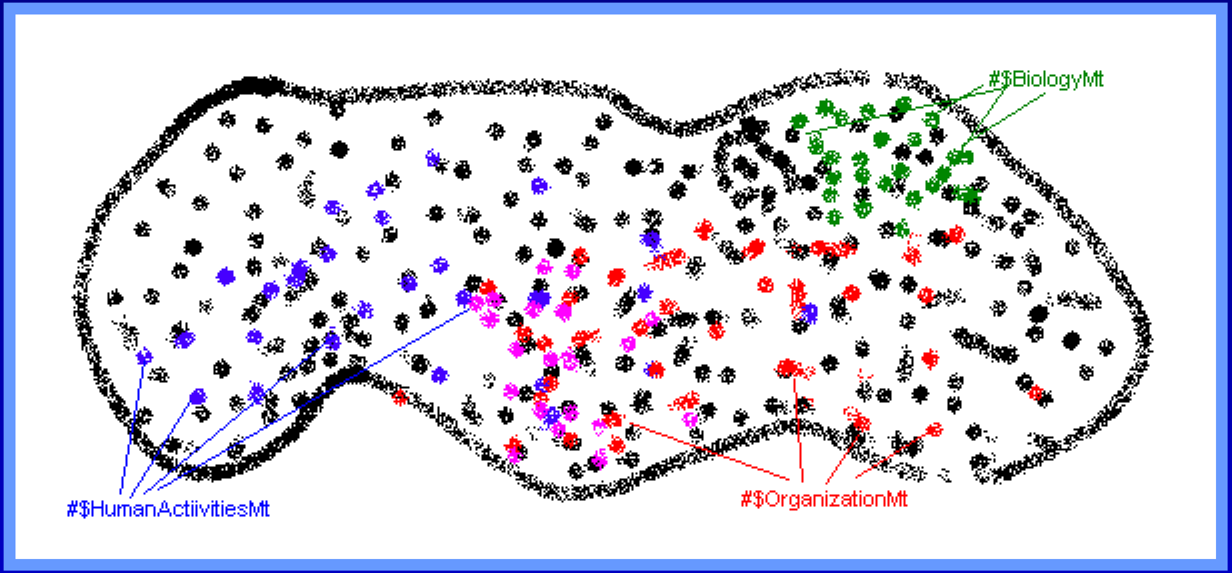
- The assertions within a microtheory must be mutually consistent
  - no **monotonic** contradictions allowed within a single microtheory
- Assertions in different microtheories may be inconsistent

in MT1: tables, etc., are solid
in MT2: tables are mostly space

in MT1: Mandela is an elder statesman
in MT2: Mandela is President of South Africa
in MT3: Mandela is a political prisoner

CYCORP

Every Assertion is in a Microtheory



- Every assertion falls within at least one microtheory
- Currently, every microtheory is a reified (named) term, such as #$HumanActivitiesMt or #$OrganizationMt
- Mts are one way of indexing all the assertions in Cyc

CYCORP

Why Have Microtheories?

Better/faster/more scalable knowledge base building
Better/faster/more scalable *inferencing,* too.

- To focus development of the Cyc knowledge base
- To enable shorter and simpler assertions

Mandela is president
vs. Mandela is president throughout 1995 in South Africa

Tables are solid
vs. At granularity usually considered by humans, tables are solid

- To cope with global inconsistency in the KB, inevitable at this scale
  - Each mt is locally consistent (content in unrelated mts is not visible)
  - Good for handling divergence (different points of view, scientific theories, changes over time)

CYCORP

Why Have microtheories? (cont.)

Better/faster/more scalable knowledge base building
Better/faster/more scalable *inferencing,* too.

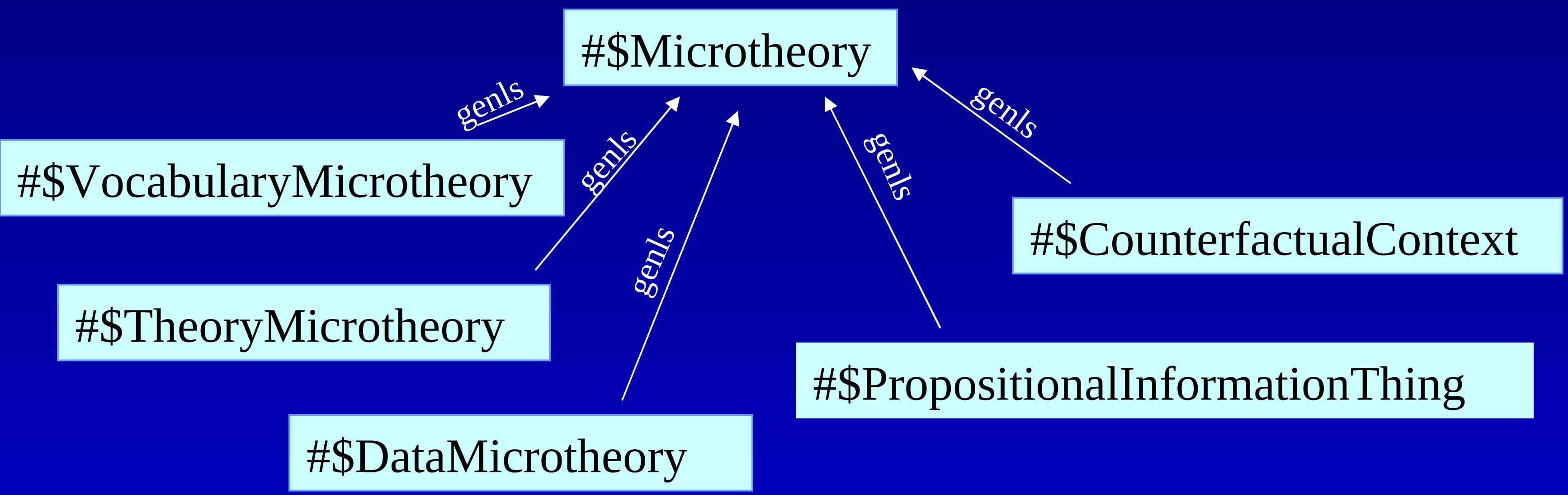
- To focus development of the Cyc knowledge base
- To enable shorter and simpler assertions

Mandela is president
vs. Mandela is president throughout 1995 in South Africa

Tables are solid
vs. At granularity usually considered by humans, tables are solid

- To cope with global inconsistency in the KB, inevitable at this scale
  - Each mt is locally consistent (content in unrelated mts is not visible)
  - Good for handling divergence (different points of view, scientific theories, changes over time)

Some types of microtheories



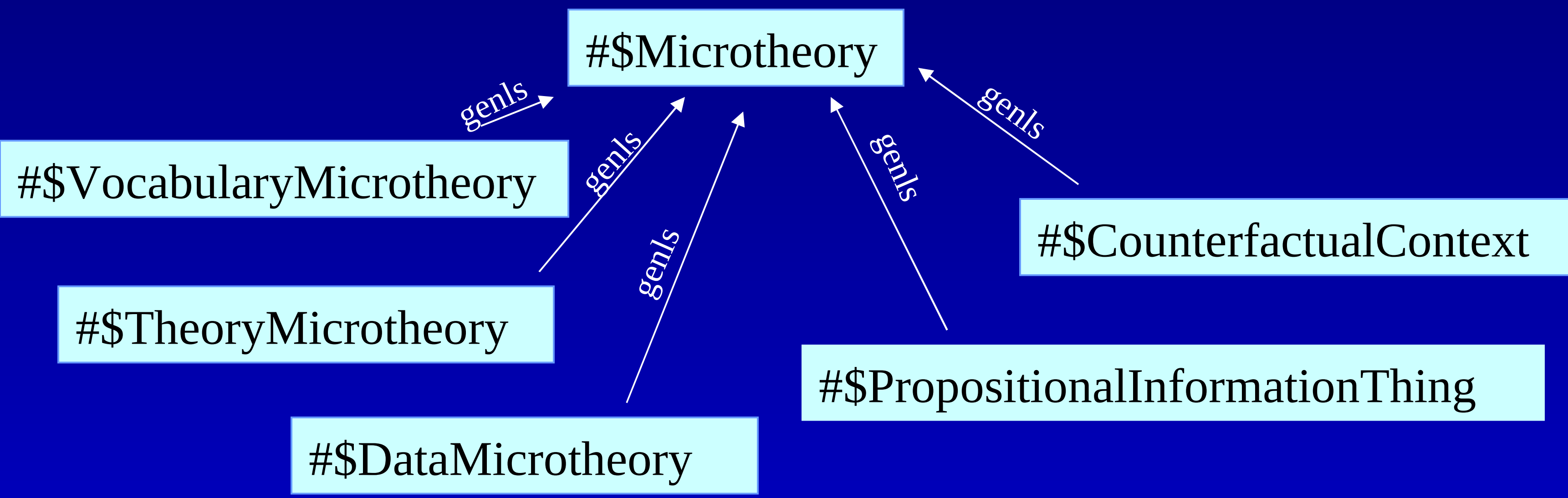
- #$VocabularyMicrotheory -- each instance contains definitions of general concepts used in a knowledge domain
- (e.g., #$TransportationVocabMt, #$ComputerSoftwareVocabMt)

- #$TheoryMicrotheory -- each instance contains general assertions in a knowledge domain (e.g., #$TransportationMt ,#$ComputerSoftwareMt).

- #$DataMicrotheory -- each instance contains assertions about specific individuals (e.g., #$TransportationDataMt, #$ComputerSoftwareDataMt)

CYCORP

Some types of microtheories



- #$PropositionalInformationThing --each instance of this collection contains assertions representing the propositional content of some #$InformationBearingThing (such as a picture, text, or database table).

- #$CounterfactualContext -- each instance of this collection contains at least one assertion which is not generally taken to be true in the real world (e.g., #$TheSimpsonsMt, #$SQ77bMt)

Microtheory predicates: #$ist

Explicitly relates a microtheory to a formula that is true in that microtheory.

- (#$ist MT FORMLA) means that the Cyc formula FORMLA is true in the microtheory MT.

```
(#$ist #$CyclistsMt (#$isa #$Lenat #$Person))

(#$ist #$NaiveStateChangeMt
  (#$implies
    (#$and
      (#$isa ?FREEZE #$Freezing)
      (#$outputsCreated ?FREEZE ?OBJ))
    (#$stateOfMatter ?OBJ #$SolidStateOfMatter)))
```

CYCORP

Microtheory predicates: #$genlMt

Relates two microtheories such that one of them inherits the assertions in the other; i.e., the first microtheory has access to the assertions in the second microtheory.

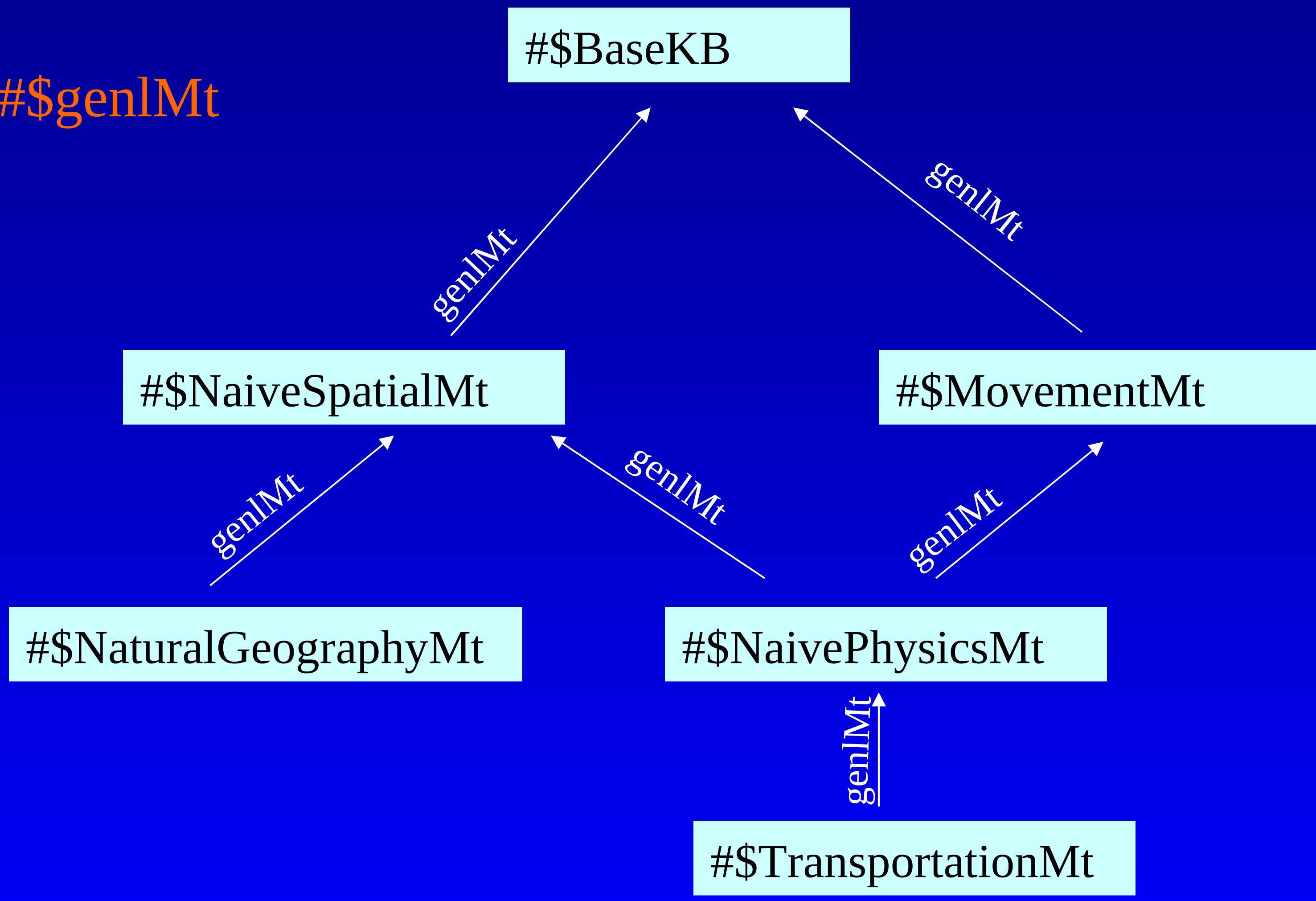
- (#$genlMt MT-1 MT-2) means that every assertion which is true in MT-2 is also true in MT-1.
- #$genlMt is transitive.

- **(#$genlMt #$TransportationMt #$NaivePhysicsMt)**
- **(#$genlMt #$ModernMilitaryTacticsMt #$ModernMilitaryVehiclesMt)**
- **(#$genlMt #$EconomyMt #$TransportationMt)**

CYCORP

Microtheory predicates, cont'd.

#$genlMt



Finding the right microtheory

- Microtheory placement is important in both making assertions and asking queries.
  - An assertion is visible in all and only the mts that inherit from the mt in which it is placed.
  - A query is answered using all and only assertions in mts visible from the mt in which it is asked.
  - Well-formedness requires visibility of definitional information (#$isa, #$genls, #$arity, #$arg1Isa . . .) for all the terms used.

- Good placement of assertions makes them visible in the microtheories in which they are needed. That is, good placement is not too specific.

- Good placement of assertions makes them invisible in microtheories in which they are not needed; otherwise, search space for inference in the lower microtheories is needlessly increased. That is, good placement is not too general.

CYCORP

Finding the right microtheory

- 3607 instances of #$Microtheory as of 02/05/01
  - 254 instances of #$GeneralMicrotheory
- For now, no substitute for familiarity
- To determine the function of some microtheory:
  - read the #$comment
  - examine the assertions
  - examine the place in the mt hierarchy in which it fits
  - if still unclear, consult the #$myCreator (or someone who has made many assertions in that microtheory)

CYCORP

Forthcoming Changes/Improvements

For more efficient ontology building and inference, we want:

- Dynamic generation of microtheories
- Software "power tools" that suggest best microtheory placement for an assertion or a query
- More targeted, smaller contexts, i.e., re-place each assertion

- These, in turn, require:
  - More explicit representation of context features e.g., topic, level of granularity, time period in which it holds,…
  - More explicit representation of the relationships that hold between contexts (besides just #$genlMt)

These improvements are part of :

* **RKF tools**
* **Context overhaul**

Summary

- What is a microtheory?
- Why have microtheories?
- Some types of microtheories
- Microtheory predicates
- Finding the right microtheory

CYCORP