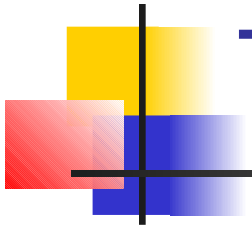
Ontology primer

Playing with RacerPro



TOC

- Why Ontology?
- What is Ontology?
- Introducing RacerPro
  - Language definitions
- Difficult Ideas
- Building Knowledge Bases
- Racer Query Demo

Why Ontology?

Semantic-less Web

- Everyone is familiar with Google – and understands how Google's keyword search works.
- Google searches can be surprisingly frustrating – too often, too much information comes back – most of which is just wrong.
- Google may be the first example of a very highly-valued company whose products mostly return the wrong answer!

Electronic agents cannot search…

- OWL and OWL and OWL
- Same
  - Same (gleich) car vs. Same (selbe) school
- FOL
  - (<- ancestor(?x ?y) (parent ?x ?y))
    (<- ancestor(?x ?z) (parent ?x ?y)
    (ancestor ?y ?z))

Goals

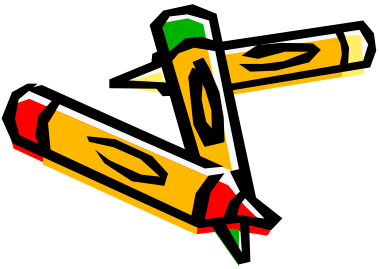
- Share common understanding of the structure of information among people or software agents
- Enable reuse of domain knowledge
- ….
- Software agents use ontologies and KBs build from ontologies as data

Query with semantics

- Competency queries
  - Which wine characteristics should I consider when choosing a wine?
  - Is Bordeaux a red or white wine?
  - Does Cabernet Sauvignon go well with seafood?
  - What is the best choice of wine for grilled meat?
  - Which characteristics of a wine affect its appropriateness for a dish?
  - Does a bouquet or body of a specific wine change with vintage year?
  - What were good vintages for Napa Zinfandel?

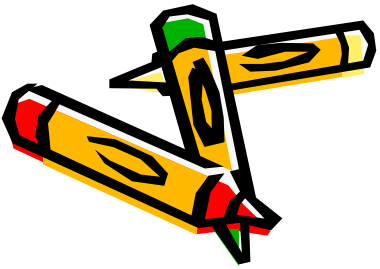
What is Ontology?

- Knowledge Representation based on *Description Logics*
  - Machine-interpretable definitions of basic concepts in the domain and relations among them



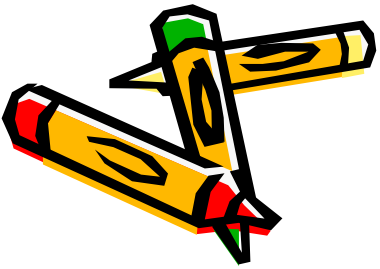
What are Description Logics?

- A family of logic based Knowledge Representation formalisms
  - Descendants of semantic networks and KL-ONE
  - Describe domain in terms of concepts (classes), roles (properties, relationships) and individuals.



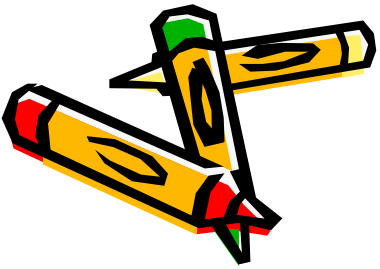
Why Description Logic?

- 20+ years of DL research
  - Well defined semantics
  - Formal properties well understood
    - I can't find an efficient algorithm, but neither can all these famous people.
  - Known reasoning algorithms
  - Highly optimized implemented systems



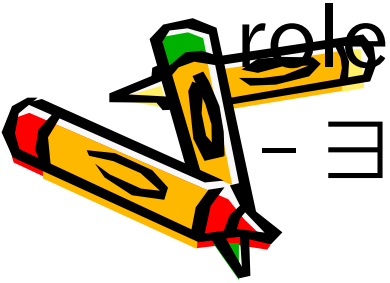
What is Ontology again?

- *There is Nothing New under the Sun*
  - "An Overview of the KL-ONE Knowledge Representation System", R.J. Brachman and J. Schmolze, Cognitive Sci 9(2), 1985.
  - "Semantic Nets" were first invented for computers by Richard H. Richens in 1956 for machine translation of natural languages. They were developed by Robert F. Simmons in the early 1960s and later featured prominently in the work of M. Ross Quillian in 1966.
  - Syntax (XML) is LEAST important



DL Basics

- Concepts
  - Person, Female, Woman, Mother
- Roles
  - hasChild, loves
- Individuals
  - Charles, Doris, Eve
- Operators (for forming concepts and roles)
  - $\exists \forall$

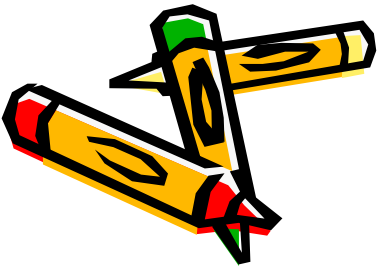


The DL Family (1)

- ***ALC***
  - Concepts constructed using booleans
    - $\cap$, $\cup$, $\neg$
    - $\exists$, $\forall$
  - Only atomic roles

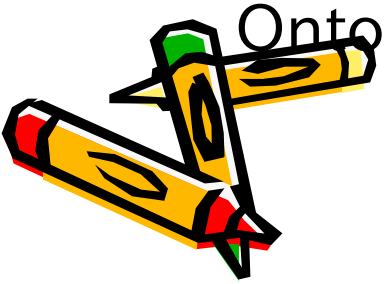
E.g., Person all of whose children are either Doctors or have a child who is Doctor:

$$\text{Person} \cap \forall \text{hasChild}.(\text{Doctor} \cup \exists \text{hasChild}.\text{Doctor})$$



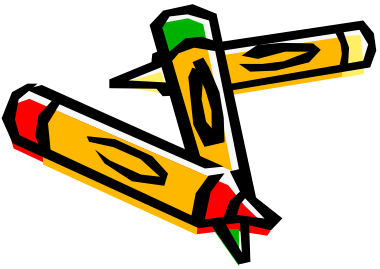
The DL Family (2)

- **ALC**
  - *S* often used for *ALC*
- Additional letters
  - *H* for role hierarchy
  - *O* for nominals/singleton classes
  - *I* for inverse roles
  - *Q* for qualified number restrictions
  - *N* for number restrictions
- **SHOIN** is the basis for W3C's OWL Web Ontology Language



DL Knowledge Base

- T-BOX
  - Conceptual knowledge representation
- A-BOX
  - Knowledge about the instances of a domain



What is RacerPro?

- Renamed ABox Concept Expression Reasoner Professional
  - *ALCQH I r+(D)-*
  - Managing semantic web ontologies based on OWL
  - Semantic web information repository with optimized retrieval engine

RacerPro Knowledge Bases

RacerPro concept terms

- C →
  - CN
  - *top* *bottom*
  - (not c) (and c1 … cn) (or c1 … cn)
  - (some R C) (all R C)
  - (at-least n R) (at-least n R C)
  - (at-most n R) (at-most n R C)
  - (exactly n R) (exactly n R C)
  - (a AN) (an AN) (no AN)
  - CDC

RacerPro concept terms

- CDC →
  - (min AN integer) (max AN integer) (equal AN integer) (equal AN AN)
  - (divisible AN cardinal) (not-divisible AN cardinal)
  - (> aexpr expr) (>= aexpr aexpr) (< aexpr aexpr) (<= aexpr aexpr) (<> aexpr aexpr) (= expr aexpr)
  - (string= AN string) (string<> AN string) (string= AN AN) (string<> AN AN)
- string →
  - " letter* "
- aexpr →
  - AN
  - real
  - (+ aexpr1 aexpr1 *) …

RacerPro role terms

- R →
  - RN
  - (inv RN)

Concept Axioms and T-boxes

- (implies C1 C2)
- (equivalent C1 C2)
- (disjoint C1 … Cn)
- (define-primitive-concept CN C)
- (define-concept CN C)

Role Declarations

- (define-primitive-role RN &key …)
  - :transitive
  - :feature
  - :symmetric
  - :reflexive
  - :inverse
  - :domain
  - :range
  - :parents

```
(define-primitive-role |wine#madeFromGrape|
    :parents (|food#madeFromFruit|)
    :inverse |wine#madeIntoWine|
    :domain |wine#Wine| :range |wine#WineGrape|)

(define-primitive-attribute |wine#hasBody| ; feature t
    :parents (|wine#hasWineDescriptor|)
    :range |wine#WineBody|)
```

```
(define-concept |wine#WhiteBordeaux|
                (and |wine#Bordeaux| |wine#WhiteWine|))
```

```
(define-primitive-concept |wine#Wine|)

(implies |wine#Wine|
        (exactly 1 |wine#hasMaker|) )

(implies |wine#Wine|
        (some |wine#locatedIn| |wine#Region|))
.....
```

```
(define-concept |wine#Meritage|
    (and |wine#Wine|
        (all |wine#madeFromGrape|
            (or |wine#CabernetSauvignonGrape|
                |wine#CabernetFrancGrape| |wine#MalbecGrape|
                |wine#PetiteVerdotGrape| |wine#MerlotGrape|))
        (at-least 2 |wine#madeFromGrape|)))

(implies |wine#Meritage| (some |wine#hasColor| |wine#Red|))
```

```
(disjoint |food#Shellfish| |food#Fish|)


(equivalent full-or-medium-bodied-wine
    (and |wine#Wine|
        (all |wine#hasBody|
          (or |wine#Full| |wine#Medium|))))
```

Individual Assertions and A-boxes

- (instance IN C)
- (related IN1 IN2 R)
- (constrained ON IN R)
  - (constrained document-1 isbn-1 isbn)
- (constraints CDC)
  - (constraints (equal isbn-1 1234567))

```
(instance |wine#WhitehallLaneCabernetFranc|

          |wine#CabernetFranc|)

(related |wine#WhitehallLaneCabernetFranc|
         |wine#Medium| |wine#hasBody|)
(related |wine#WhitehallLaneCabernetFranc|
         |wine#Moderate| |wine#hasFlavor|)
(related |wine#WhitehallLaneCabernetFranc|
         |wine#Dry| |wine#hasSugar|)
(related |wine#WhitehallLaneCabernetFranc|
         |wine#WhitehallLane| |wine#hasMaker|)
(related |wine#WhitehallLaneCabernetFranc|
         |wine#NapaRegion| |wine#locatedIn|)
```

Consistency check

- tbox-coherent-p
- check-tbox-coherence
- abox-consistent-p
- check-abox-coherence

Knowledge Base Inferences

- Completion
- Classification/Subsumption
- Query and Rule application

```
(define-concept full-or-medium-bodied-wine
   (and |wine#Wine|
        (all |wine#hasBody|
             (or |wine#Full| |wine#Medium|))))
  → T

(define-concept medium-or-light-bodied-wine
   (and |wine#Wine|
        (all |wine#hasBody|
                  (or |wine#Medium| |wine#Light|))))
  → T

(define-concept special-bodied-wine
   (and full-or-medium-bodied-wine
                medium-or-light-bodied-wine))
  → T
```

```
(concept-subsumes? medium-or-light-bodied-wine
                   special-bodied-wine)
```

→ T

```
(concept-subsumes? medium-or-light-bodied-wine
                   (and |wine#Wine|
                     (all |wine#hasBody| |wine#Medium|)))
```

→ T

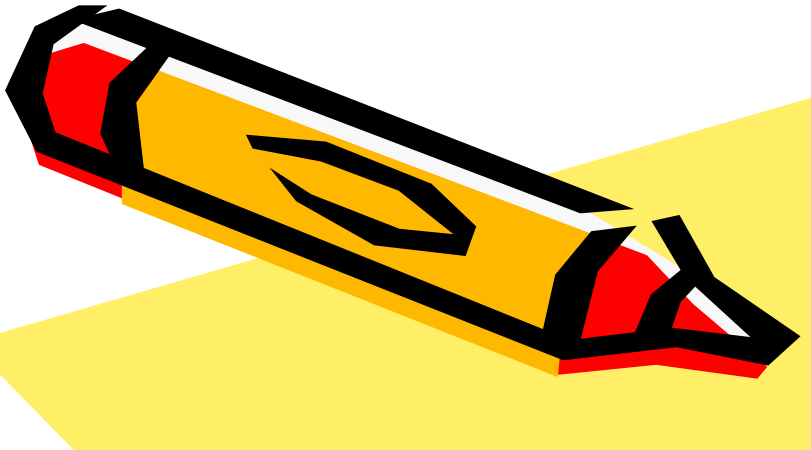
```
;; (retrieve (?x) (?x (and |wine#Wine|
;;                         (all |wine#hasBody| |wine#Medium|))))
(retrieve (?x) (?x special-bodied-wine))
 → (((?X |wine#ClosDeLaPoussieSancerre|))
    ((?X |wine#ChateauDYchemSauterne|))
    ((?X |wine#WhitehallLaneCabernetFranc|))
    ((?X |wine#VentanaCheninBlanc|))
    ((?X |wine#StonleighSauvignonBlanc|))
    ((?X |wine#SelaksSauvignonBlanc|))
    ((?X |wine#SelaksIceWine|))
    ((?X |wine#SaucelitoCanyonZinfandel1998|))
    ((?X |wine#SaucelitoCanyonZinfandel|))
    ((?X |wine#PulignyMontrachetWhiteBurgundy|)) ...)
```

Knowledge Base Operations

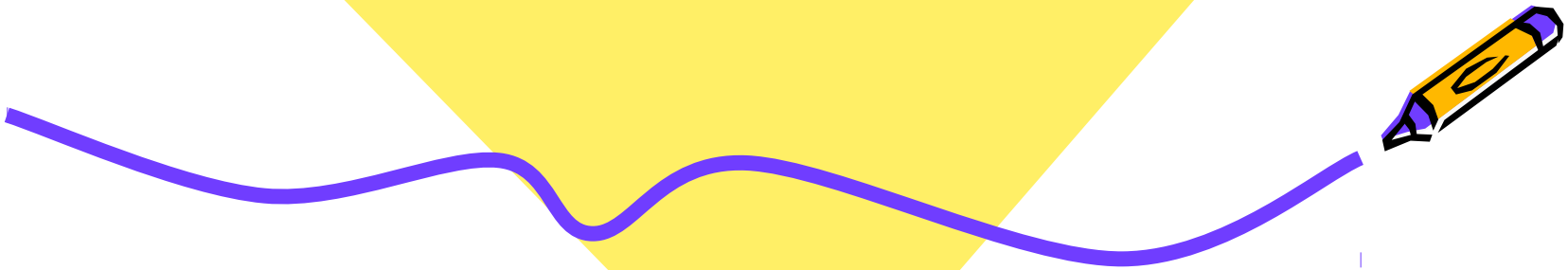
- What are all the instances of this concept? (which individuals satisfy this description?)
- Which concepts does this individual satisfy?
- Which individuals fill role **R** on individual **I**?
- How is role **R** restricted on concept **C** (or on individual **I**)?

```
(concept-instances |wine#CabernetFranc|)
 → (|wine#WhitehallLaneCabernetFranc|)

(concept-instances
                (and
                    (some |wine#hasBody| |wine#Full|)
                    (some |wine#hasColor| |wine#Red|)
                    (some |wine#locatedIn|
                                |wine#CaliforniaRegion|)))
 → (|wine#SeanThackreySiriusPetiteSyrah|
    |wine#SantaCruzMountainVineyardCabernetSauvignon|
    |wine#MountEdenVineyardEstatePinotNoir|
    |wine#ElyseZinfandel|
    |wine#CotturiZinfandel|)
```

```
(individual-types |wine#WhitehallLaneCabernetFranc|)
  → ((|wine#CabernetFranc|)
     (|wine#RedTableWine| |wine#DryRedWine|)
     (|wine#DryWine| |wine#TableWine|)
     (|food#Wine| |wine#Wine|)
     (|food#PotableLiquid|) (|food#ConsumableThing|)
     (*TOP* TOP) (|wine#RedWine|)
     (|wine#AmericanWine|) (|wine#CaliforniaWine|))

(individual-fillers |wine#WhitehallLaneCabernetFranc|
                    |wine#hasFlavor|)
  → (|food#Moderate|)

(individual-filled-roles |wine#WhitehallLaneCabernetFranc|
                         |wine#WhitehallLane|)
  → (|wine#hasMaker|)
```



Difficult Ideas

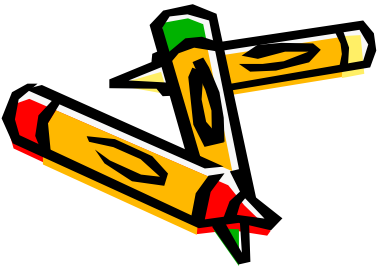


Primitive and Defined Concepts

- Primitive concept is appropriate when no complete definition exists or when only part of a completely known definition is relevant.
- Defined concepts are appropriate when the complete definition is known and relevant, or when one wants the system to determine membership in a class.

CONCEPT-IS-PRIMITIVE-P

```
;; non-primitive (definitional) examples
(define-concept |wine#CaliforniaWine|
   (and |wine#Wine|
        (some |wine#locatedIn| |wine#CaliforniaRegion|)))

(define-concept |wine#WhiteWine|
   (and |wine#Wine|
        (some |wine#hasColor| |wine#White|)))

;; primitive (non-definitional) examples
(implies |wine#Port| |wine#RedWine|)
(implies |wine#Port|
     (some |wine#locatedIn| |wine#PortugalRegion|))
....

(implies |wine#Vintage|
         (and (at-least 1 |wine#hasVintageYear|)
              (at-most 1 |wine#hasVintageYear|)))
```



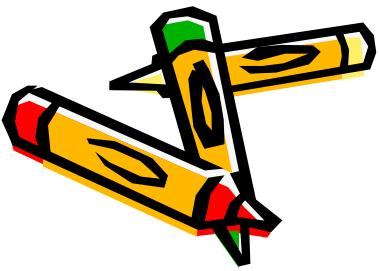
Definitional and Incidental Properties

```
(define-concept |wine#Zinfandel|
    (and |wine#Wine|
        (some |wine#madeFromGrape| |wine#ZinfandelGrape|)
        (at-most 1 |wine#madeFromGrape|)))

(implies |wine#Zinfandel| (some |wine#hasColor| |wine#Red|))
(implies |wine#Zinfandel| (some |wine#hasSugar| |wine#Dry|))
(implies |wine#Zinfandel|
            (all |wine#hasBody|
                (or |wine#Full| |wine#Medium|)))
(implies |wine#Zinfandel|
                (all |wine#hasFlavor|
                    (or |wine#Moderate| |wine#Strong|)))
```

Concepts and Individuals

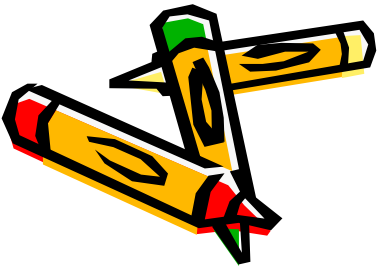
- Individuals have unique identities and are countable.
- Concepts are descriptions and the concept space is infinite.
- Individuals can change.
- Concept definitions and their relationships to each other do not change.

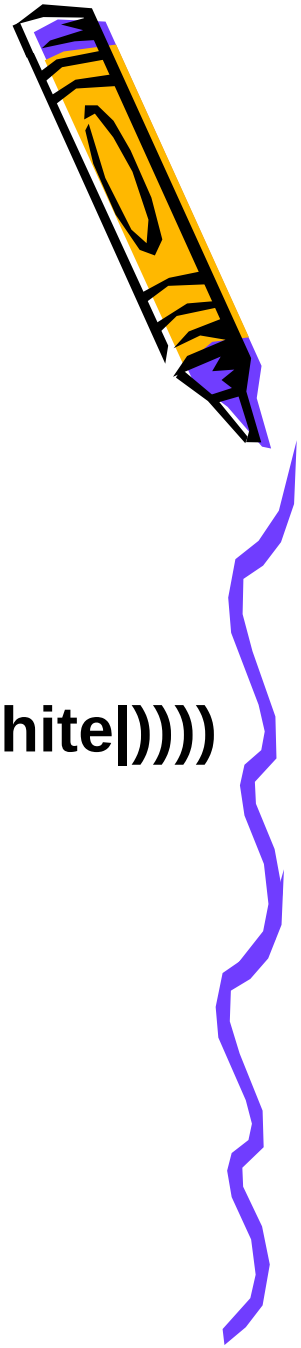
```
(define-concept picnic-basket
    (and basket
        (exactly 2 |food#hasDrink|)
        (all |food#hasDrink| |wine#Wine|)
        (at-least 3 |food#hasFood|)
        (all |food#hasFood| |food#EdibleThing|)))

(define-concept california-picnic-basket
    (and picnic-basket
          (all |food#hasDrink| |wine#CaliforniaWine|)))

(define-concept kalin-cellars-basket
    (and picnic-basket (some |food#hasDrink|
                             (or |wine#KalinCellarsChardonnay|
                                 |wine#KalinCellarsCabernet|)))
```



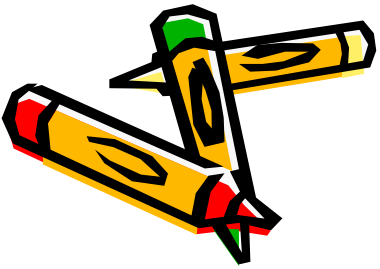
Rule (Query) Application

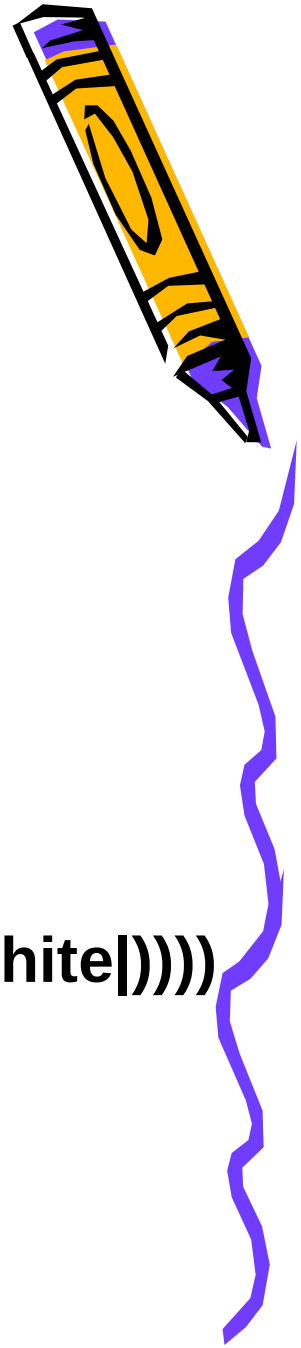
- A rule is not actually “fired” until an individual is found to be an instance of the antecedent concept.





```
....
(implies |food#SeafoodCourse|
        (all |food#hasDrink|
            (some |wine#hasColor| |food#White|)))
....

(retrieve (?x) (?x (some |food#hasDrink|
                         (some |wine#hasColor| |food#White|))))
 → NIL
(retrieve (?x) (?x |food#SeafoodCourse|))
 → NIL
```

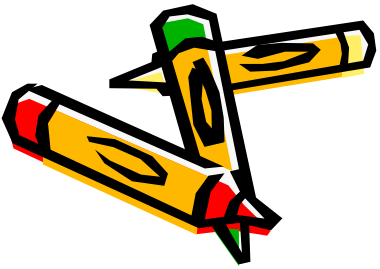




....
(implies |food#SeafoodCourse|
        (all |food#hasDrink|
            (some |wine#hasColor| |food#White|)))
....

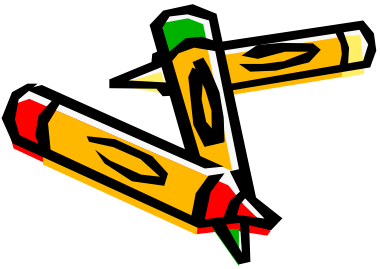
(instance sea-food |food#SeafoodCourse|)
 → T

(retrieve (?x) (?x (some |food#hasDrink|
                         (some |wine#hasColor| |food#White|))))
 → (((?X SEA-FOOD)))



Features and Roles

- Features (also called attributes)
  - Restrict a role to be a functional role, e.g. each individual can only have up to one filler for this role.



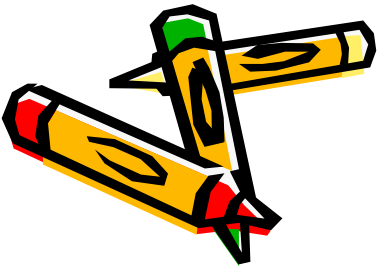
Open-World and Closed-World Assumption

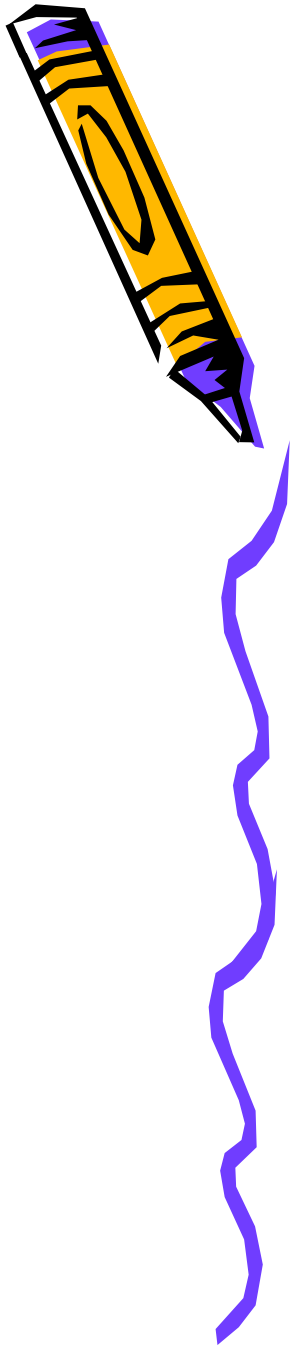
(retrieve (?x ?y) (?x ?y |wine#madeFromGrape|))

→ (((?X |wine#ChateauDYchemSauterne|)
(?Y |wine#SemillonGrape|))
((?X |wine#ChateauDYchemSauterne|)
(?Y |wine#SauvignonBlancGrape|)))

(retrieve () (|wine#ChateauDYchemSauterne|
(at-least 2 |wine#madeFromGrape|)))

→ NIL





```
(all-different (|wine#SauvignonBlancGrape|
                |wine#SemillonGrape|))
→ T

(retrieve () (|wine#ChateauDYchemSauterne|
    (at-least 2 |wine#madeFromGrape|)))
→ T

(retrieve () (|wine#ChateauDYchemSauterne|
    (at-most 2 |wine#madeFromGrape|)))
→ NIL

(retrieve () (|wine#ChateauDYchemSauterne|
    (exactly 2 |wine#madeFromGrape|)))
→ NIL
```

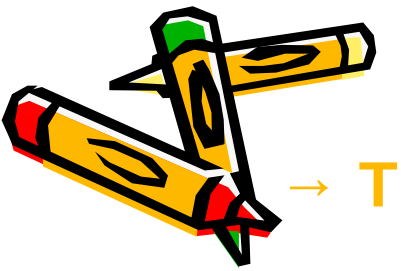
Classic Negation vs. Negation as Failure

- Two types of "negative" situations with respect to a goal G:
  - being able to solve the goal ¬ G; or
  - being unable to solve the goal G.

```
(retrieve () (|wine#WhitehallLaneCabernetFranc|
                (not |wine#WhiteWine|)))
```

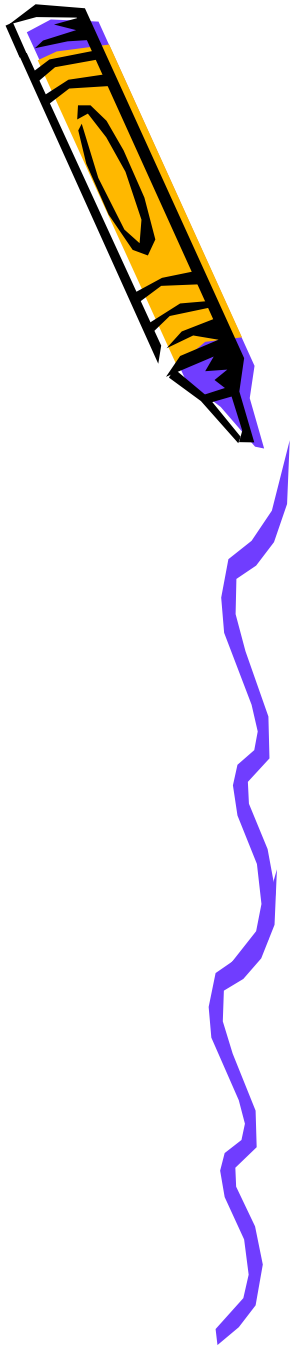
→ NIL

```
(retrieve ()
    (neg (|wine#WhitehallLaneCabernetFranc|
            |wine#WhiteWine|)))
```



→ T

Multiple Inheritance



(instance |wine#StGenevieveTexasWhite|
|wine#WhiteWine|)

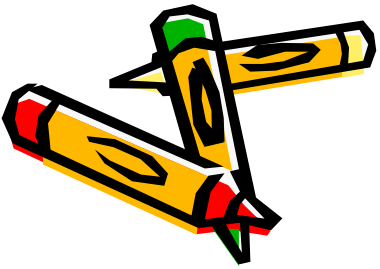
→ T

(instance |wine#StGenevieveTexasWhite|
|wine#RedWine|)

→ T

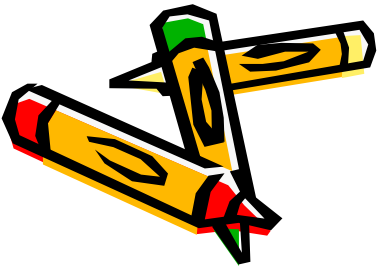
(abox-consistent-p)

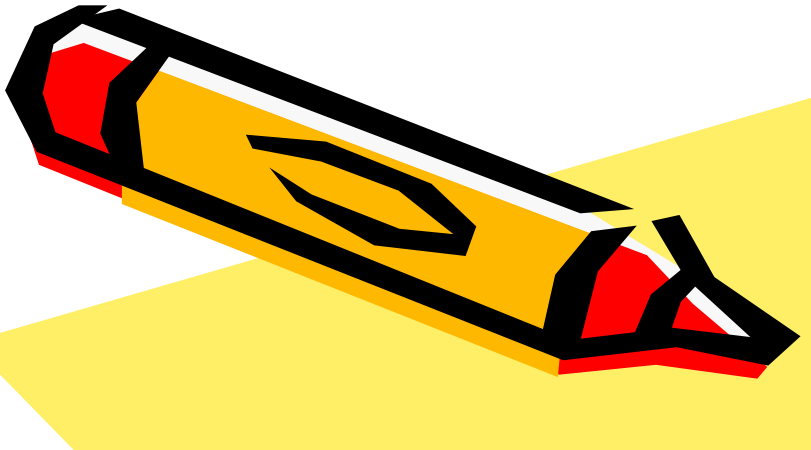
→ T



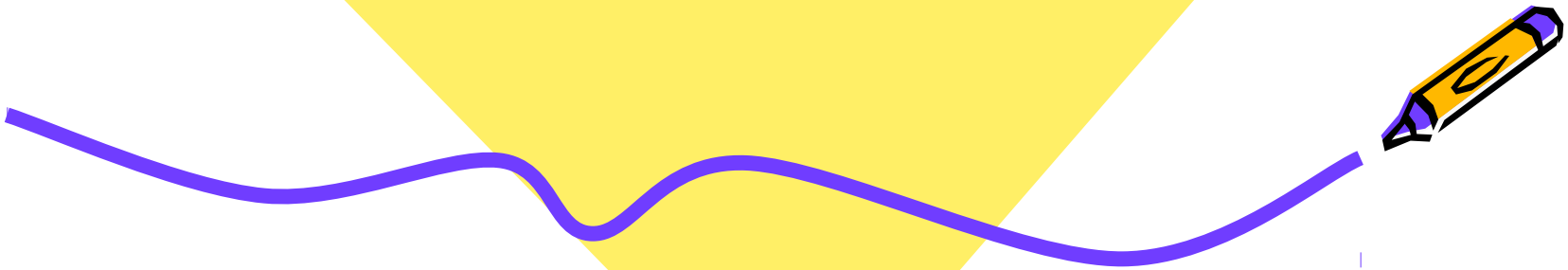
Disjoint

- Atomic Concepts vs. OOP's Classes
  - In OOP, classes are not overlapping at least unless they have an explicit common child.
  - In DL, Atomic Concepts are overlapping until disjointness axioms are entered.



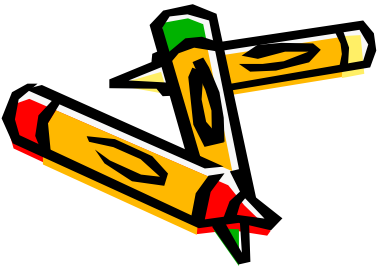


Building Knowledge Bases



Enumerate Object Types

- body, sugar, color …
- wine, grape, fish, food …
- white-wine, dry-wine …
- red-meat, seafood …
- location, winery …

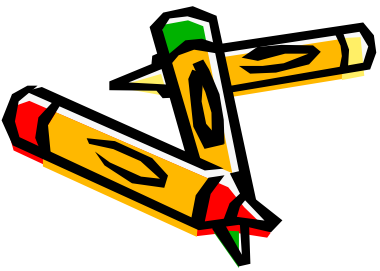


Concepts vs. Roles

- Concepts
  - Wine, Winery, Grape …
  - White-wine, Red-wine …
  - Food, Fish, Shellfish …
  - Red-meat, Seafood …
- Roles
  - **has-sugar, has-body, has-flavor, has-color** *(features/attributes)*
  - adjacent-region, made-from-grape

Hierarchy

- |food#ConsumableThing|
  - |food#PotableLiquid|
    - |wine#Wine|
      - |wine#WhiteWine|
      - |wine#RedWine|
        - |wine#RedBordeaux|
      - |wine#DryWine|
        - |wine#DryWhiteWine|
  - |food#EdibleThing|
    - |food#Seafood|
      - |food#Fish|
      - |food#Shellfish|



Individuals

(instance |wine#Red| |wine#WineColor|)
(instance |wine#Rose| |wine#WineColor|)
(instance |wine#White| |wine#WineColor|)

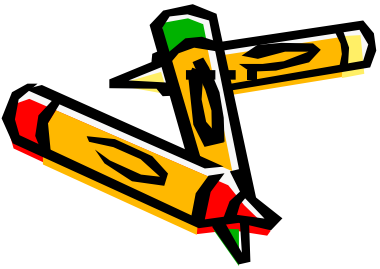
(instance |food#Crab| |food#NonOysterShellfish|)
(instance |food#Swordfish| |food#NonBlandFish|)

(instance |wine#Dry| |wine#WineSugar|)
(instance |wine#OffDry| |wine#WineSugar|)
(instance |wine#Sweet| |wine#WineSugar|)
...

Determine Properties and Parts

```
(define-primitive-attribute |wine#hasColor|
  :parents (|wine#hasWineDescriptor|)
  :domain |wine#Wine| :range |wine#WineColor|)
(define-primitive-attribute |wine#hasBody|
    :parents (|wine#hasWineDescriptor|)
    :range |wine#WineBody|)
(define-primitive-attribute |wine#hasFlavor|
    :parents (|wine#hasWineDescriptor|)
    :range |wine#WineFlavor|)
(define-primitive-attribute |wine#hasSugar|
    :parents (|wine#hasWineDescriptor|)
    :range |wine#WineSugar|)
```

```
(define-primitive-role |wine#madeFromGrape|
    :parents (|food#madeFromFruit|)
    :inverse |wine#madeIntoWine| :domain |wine#Wine|
    :range |wine#WineGrape|)
(define-primitive-role |wine#locatedIn|
    :transitive t :domain top :range |wine#Region|)
(define-primitive-role |wine#adjacentRegion|
    :inverse |wine#adjacentRegion|
    :domain |wine#Region|
    :range |wine#Region|)
```



Determine Number Restrictions.

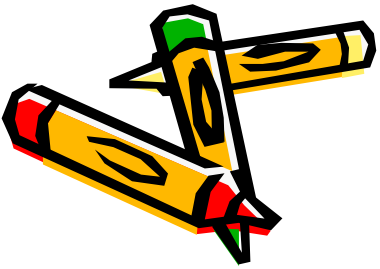
(implies |wine#Wine| (exactly 1 |wine#hasColor|))

(implies |wine#Wine| (exactly 1 |wine#hasBody|))

(implies |wine#Wine| (at-least 1 |wine#madeFromGrape|))

(implies |wine#Wine| (exactly 1 |wine#hasSugar|))

(implies |wine#Wine|
(some |wine#locatedIn| |wine#Region|))



Determine Value Restrictions

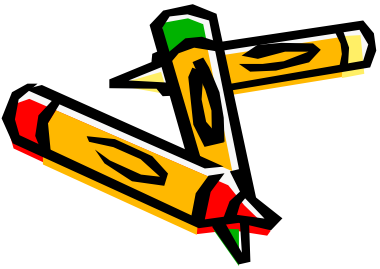
```
(define-concept |wine#WineColor|
    (or |wine#White| |wine#Rose| |wine#Red|))

(define-concept |wine#WineSugar|
    (or |wine#Sweet| |wine#OffDry| |wine#Dry|))

(define-concept |wine#WineFlavor|
    (or |wine#Delicate| |wine#Moderate| |wine#Strong|))

(define-concept |wine#WineBody|
    (or |wine#Light| |wine#Medium| |wine#Full|))
```

Distinguish Essential and Incidental Properties.

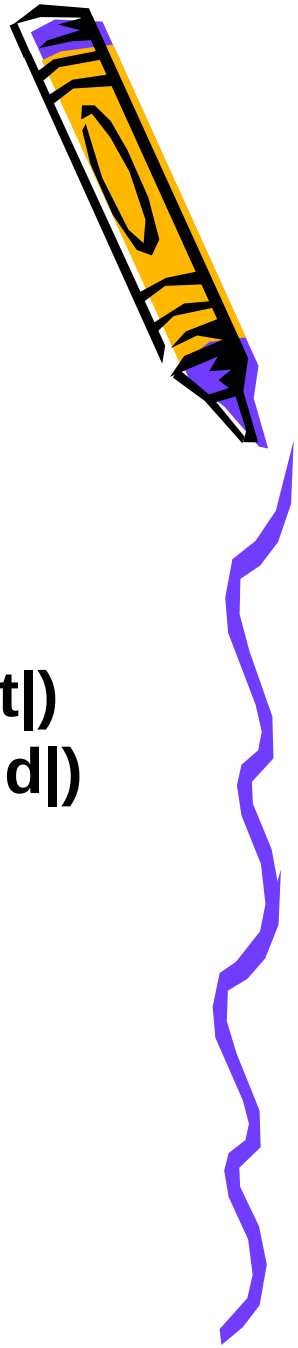
```
(define-concept |wine#Zinfandel|
    (and |wine#Wine|
        (some |wine#madeFromGrape| |wine#ZinfandelGrape|)
        (at-most 1 |wine#madeFromGrape|)))

(implies |wine#Zinfandel| (some |wine#hasColor| |wine#Red|))
(implies |wine#Zinfandel| (some |wine#hasSugar| |wine#Dry|))
(implies |wine#Zinfandel|
        (all |wine#hasBody|
            (or |wine#Full| |wine#Medium|)))
(implies |wine#Zinfandel|
        (all |wine#hasFlavor|
            (or |wine#Moderate| |wine#Strong|)))
```

Distinguish Primitive and Defined Concepts.

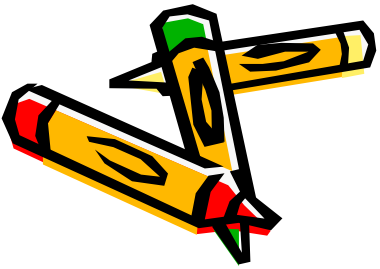
(implies |wine#Wine| |food#PotableLiquid|)
(implies |wine#Wine| (exactly 1 |wine#hasMaker|))
(implies |wine#Wine| (all |wine#hasMaker| |wine#Winery|))
(implies |wine#Wine| (at-least 1 |wine#madeFromGrape|))
(implies |wine#Wine| (exactly 1 |wine#hasSugar|))
(implies |wine#Wine| (exactly 1 |wine#hasFlavor|))
(implies |wine#Wine| (exactly1 |wine#hasBody|))
(implies |wine#Wine| (exactly 1 |wine#hasColor|))
(implies |wine#Wine| (some |wine#locatedIn| |wine#Region|))



Determine DISJOINT Concepts



(disjoint |food#Seafood| |food#Meat|)
(disjoint |food#Seafood| |food#Fowl|)
(disjoint |food#Shellfish| |food#Fish|)
(disjoint |wine#EarlyHarvest| |wine#LateHarvest|)
(disjoint |food#EdibleThing| |food#PotableLiquid|)
....
(equivalent |food#Dry| |wine#Dry|)
....



Assertions

(instance |wine#WhitehallLane| |wine#Winery|)

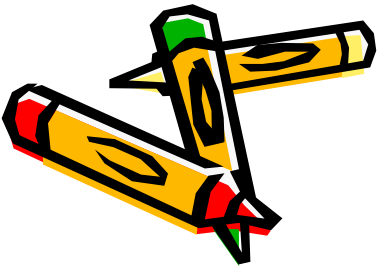
....

(related |wine#WhitehallLanePrimavera| |wine#Light|
|wine#hasBody|)

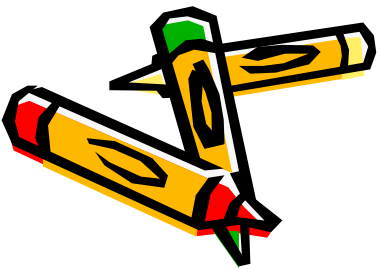
....

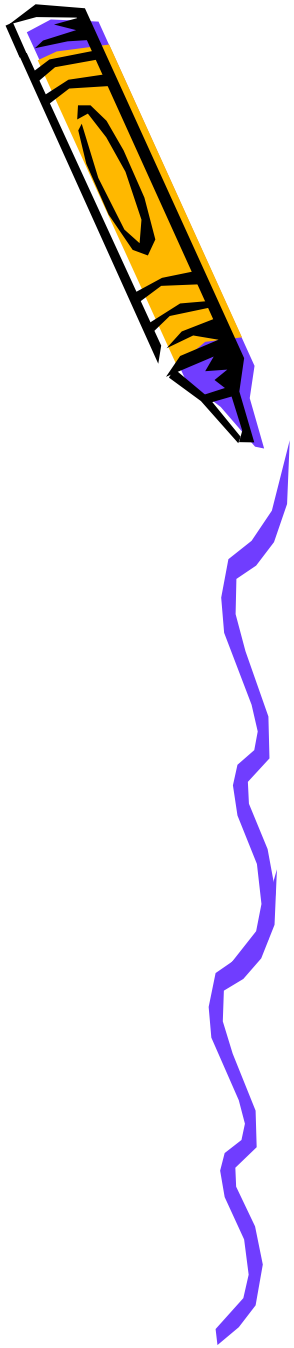
(all-different (|wine#Bancroft|
|wine#ChateauChevalBlanc| ...))

....

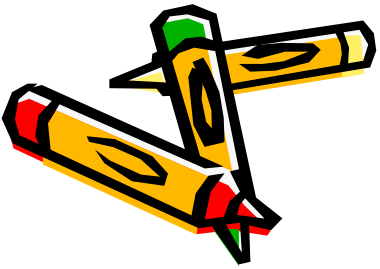


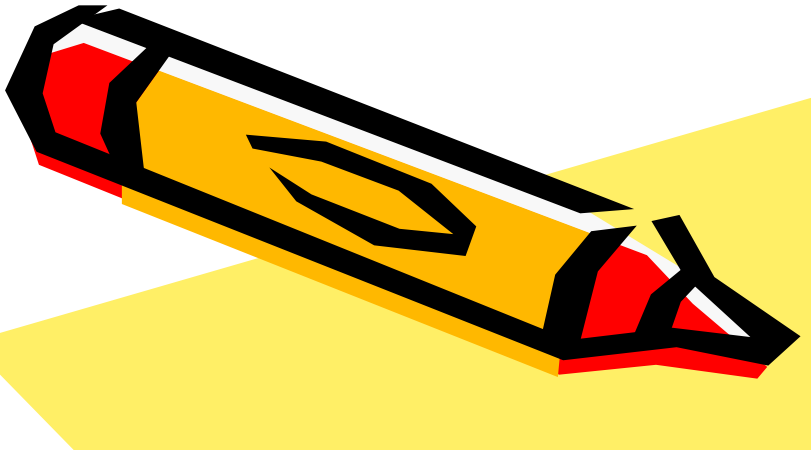
Demo



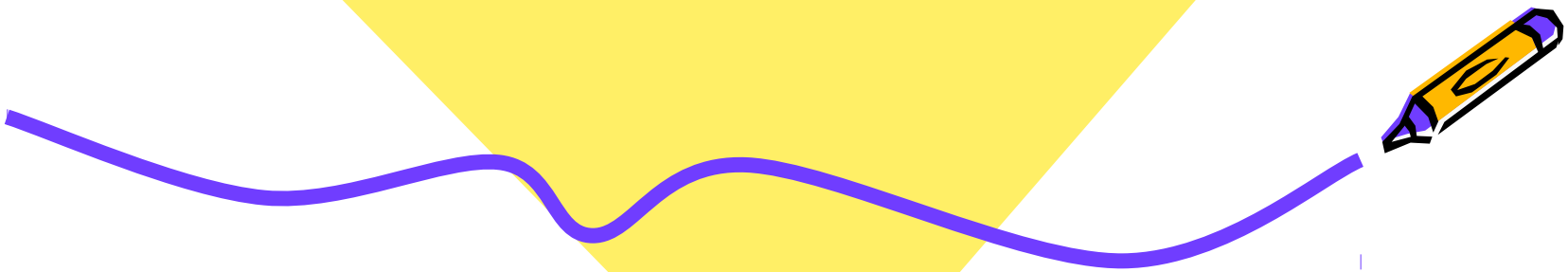


The End





付録



RacerPro Query Language nRQL

- Complex queries are build from query atoms.
  - same-as
  - and, union, neg and project-to
- Well-defined syntax and clean compositional semantics.
- *Negation as failure* as well as *true classical negation* is available.

RacerPro Query Language nRQL

- Special support for querying the *concrete domain* part of an ABox.
  - *Constraint checking* on the concrete domain
- A projection operator **project-to** for query bodies.
- Complex TBox queries are also available.

RacerPro Query Language nRQL

- Hybrid queries
  - Cost-based heuristic optimizer.
  - A facility to define queries.
  - A simple rule mechanism.
  - Queries (and rules) are maintained as objects.
  - Multi-processing of queries.
  - Support for different querying modes.

Updates

- T-box updates costs high
  - Classify
  - Normalize
- A-box and T-box files should be separated

Rules

- A rule consists of an antecedent and a consequent.
- As soon as an individual is known to satisfy the antecedent concept, the rule is “triggered,” and the individual is also known to satisfy the consequent concept.

```
(define-concept |food#SeafoodCourse|
            (and |food#MealCourse|
                 (all |food#hasFood| |food#Seafood|)))

(implies |food#SeafoodCourse|
        (all |food#hasDrink|
           (some |wine#hasColor| |food#White|)))
```

Very tricky

```
(retrieve () (|wine#WhitehallLaneCabernetFranc|
              |wine#RedWine|))
T
(retrieve () (neg (|wine#WhitehallLaneCabernetFranc|
                   |wine#RedWine|)))
T
(retrieve () (neg (|wine#WhitehallLaneCabernetFranc|
                   (not |wine#RedWine|))))
T
```

```
(retrieve (|wine#WhitehallLaneCabernetFranc|)
                  (neg (|wine#WhitehallLaneCabernetFranc|
                        |wine#RedWine|)))
((((|$?wine#WhitehallLaneCabernetFranc| |wine#SemillonGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#SauvignonBlancGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#SangioveseGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#RieslingGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#PinotNoirGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#PinotBlancGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#PetiteVerdotGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#PetiteSyrahGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#MerlotGrape|))
 ((|$?wine#WhitehallLaneCabernetFranc| |wine#MalbecGrape|)) ...)
```