A semiotic-conceptual framework for ontologies

OntoQuery - Lecture 4

Uta Priss
School of Computing,
Napier University,
Edinburgh, UK
u.priss@napier.ac.uk

May, 2004

I) Strings versus Signs

II) Peirce's Sign Triad

III) A Semiotic Conceptual Framework

Variables in maths/logic:

$x^2 + x + 15 = y$
$x = 2$
$y = 21$

true or false?

use rules for axioms, syntax, grammar

Variables in computer programs:

```
1> age := 5
2> counter := 5
3> age := age + 1
4> age := ``Golden Age''
```

equal or identical?

Line 1 and 2: equal values
Line 1 and 3: identical variables
Line 3 and 4: homographic variables

Strings versus Signs:

maths/logic	programming languages
variables are strings	variables are signs
independent of time and user single global context	depend on history and user many different contexts
equality $=$ identity	equality $\neq$ identity
set-based	array-, list-, table-based
binary	triadic

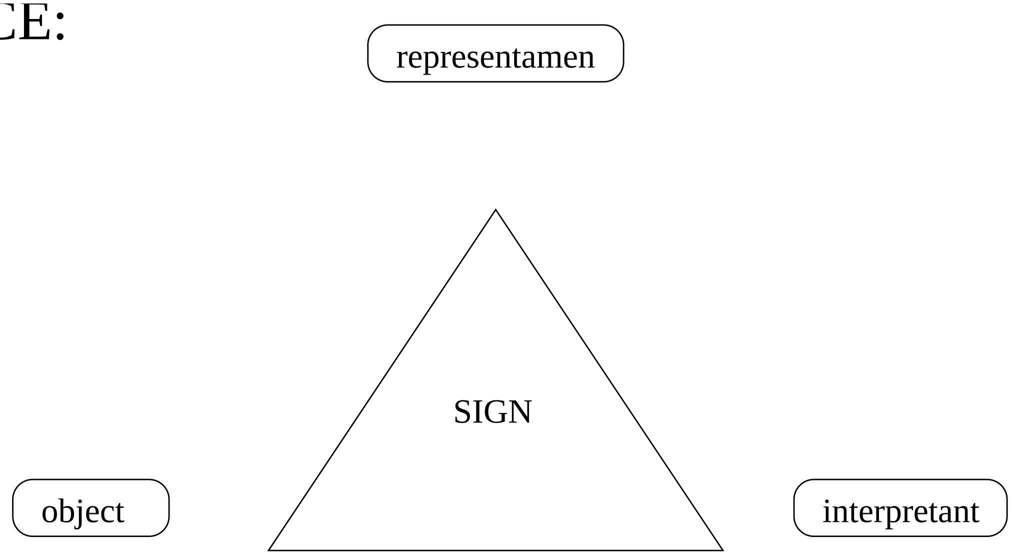
Database theory: Relational Algebra (SQL)

Contextual factors in databases:

⋆ recovery mechanisms
⋆ transaction logs
⋆ deadlocks
⋆ performance tuning
⋆ user support
⋆ versions
⋆ "error" and "exception" handling
⋆ networking and external devices

→ distractions from elegant, deterministic formal structure?

PEIRCE:



"A sign, or representamen, is something which stands to somebody for something in some respect or capacity. It addresses somebody, that is, creates in the mind of that person an equivalent sign, or perhaps a more developed sign. That sign which it creates I call the interpretant of the first sign. The sign stands for something, its object." Peirce (1897)

Variables in programming languages are triadic:

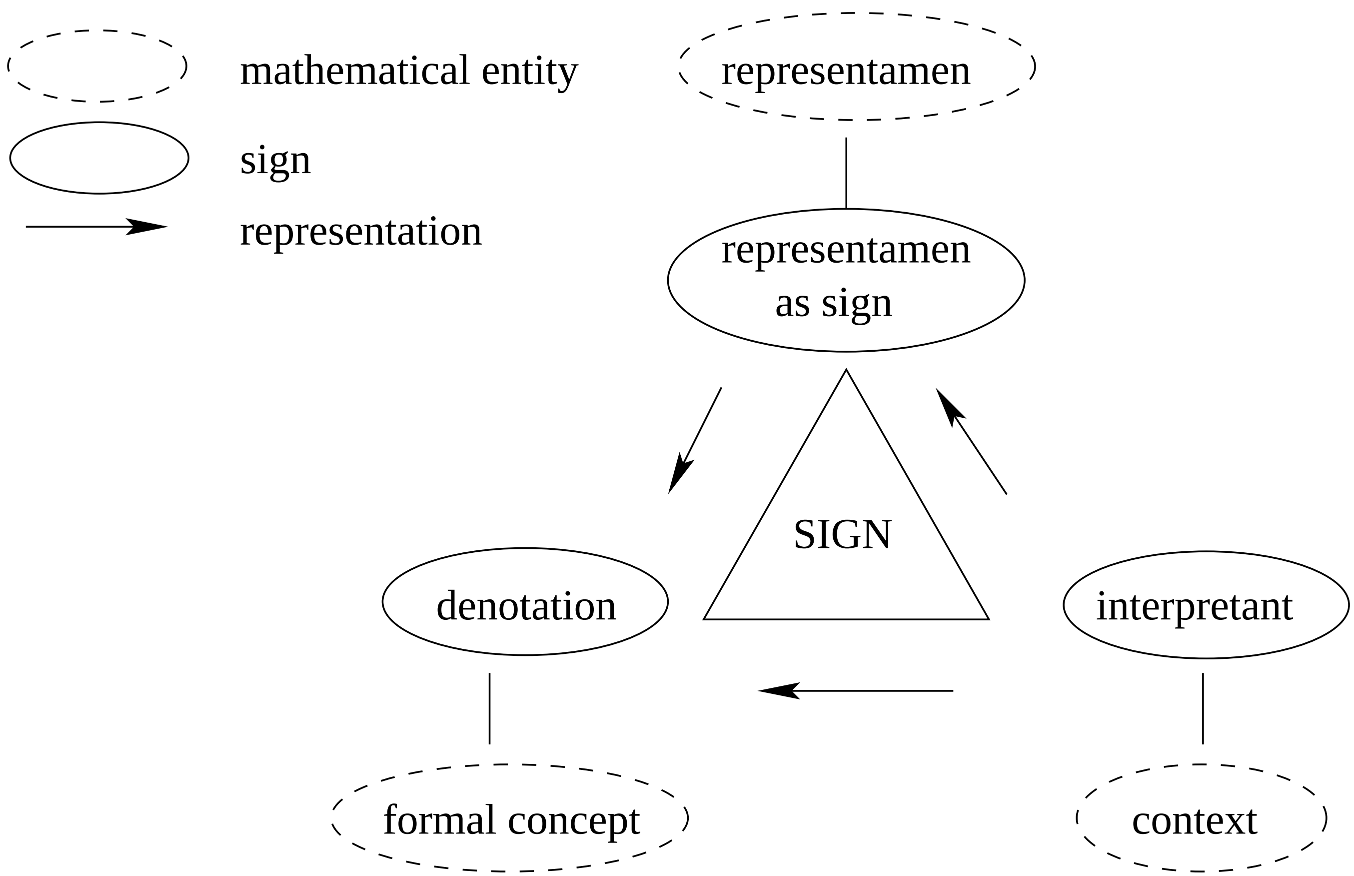
- representamen: variable name
- denotation: a type/value pair
- interpretant: operating system, language, programmer, ...

Maths/logic is binary:

- names are irrelevant (renaming does not affect content)

- content is irrelevant (formal language + grammar, proof theory)

A Semiotic Conceptual Framework:



Computational entities versus signs:

Computational entities (strings) are fully defined by axioms, rules, grammars, etc.

Signs are triadic and exist in real world interpretants.

Each of the three components of a sign can be modelled as a computational entity.

A theory which describes the combination of the three components of a sign ...

Interpretants are called **overlapping** iff they share signs.

$\longrightarrow$ Example: Uta (name), uta (Japanese for "song")
these interpretants do not overlap

Synonymy is an equivalence relation among signs which fulfills the necessary (but not sufficient) condition that synonymous signs have synonymous denotations.

$\longrightarrow$ Synonymy cannot be calculated but depends on user judgement.

Example: hot - warm - medium - cool - cold - freezing

Interpretants are called **compatible** iff signs with equal representamens are synonymous.

not compatible:	possibly compatible:
age := 5	age := 5
age := "Golden Age"	age := 6

$\longrightarrow$ compatibility is a basic condition for communication

Sign equivalences in compatible interpretants:

- Signs are **polysemous** iff they have equal representamens.

- Signs are **equal** iff they have equal representamens and equal denotations.

- Signs can be **equinymous** if they are synonymous and have equal denotations (i.e. “strong synonymy”).

- Signs are **identical** iff they have identical denotations according to some mapping called **identity** which maps signs onto identifiers.

Example:

	Paul's salary = 20K	salary 1 = 20K
PAUL's salary = 20K	equal	equinymous
Paul's salary = 25K	polysemous	synonymous

Signs are **anonymous** iff their representamens do not add any information which is not already contained in the denotations.

$\longrightarrow$ Constants in programming languages are anonymous. They have no representamens apart from their own denotations.

Compatible interpretants are **mergeable** iff all their synonyms are equinyms.

$\longrightarrow$ For anonymous signs in mergeable interpretants, all five sign equivalence relations are the same.

The difference between signs and computational entities can now be explained as follows:

Computational entities are anonymous and their meaning does not depend on special contexts.

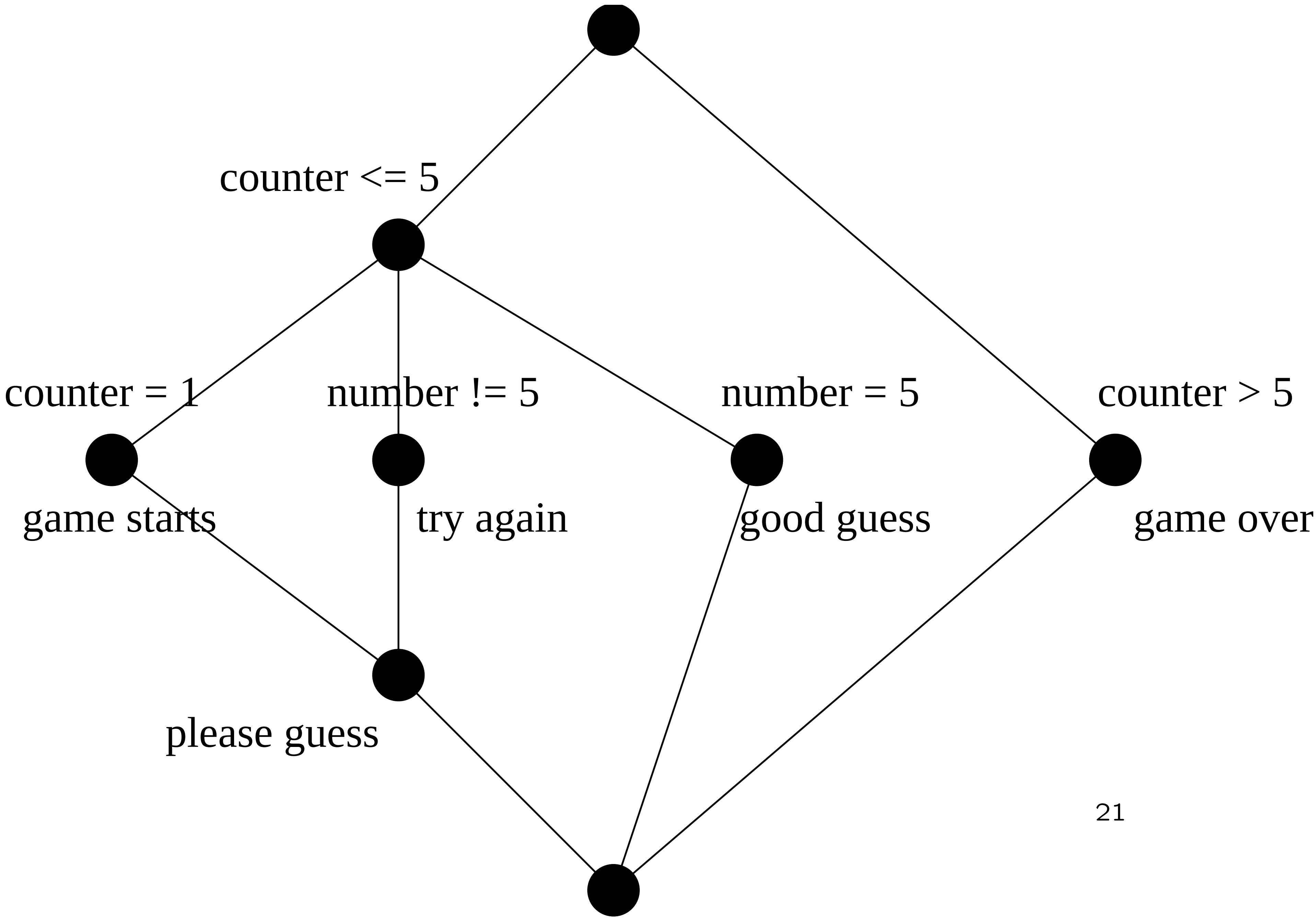
→ Signs have richer equivalence relations than computational entities. The past and present of signs can possibly be precisely modelled using computational entities, but not the future.

Sign processing implies information management tasks:

- representamen: manage names, namespaces, lexica
- denotation: manage identities and type hierarchies
- interpretants: manage contexts

Example:

```
counter = 1
print "game starts"
while counter <= 5:
    number = input("please guess the number")
    if number == 5:
        print "good guess"
        break
    else:
        print "try again"
    counter = counter +1
else:
    print "game over"
```


counter <= 5
counter = 1
number != 5
number = 5
counter > 5
game starts
try again
good guess
game over
please guess

What next?

Conceptual structures (logic, reasoning etc) are already well understood and form the foundation of a semiotic conceptual framework.

What impact do the sign equivalences have on conceptual structures?

$\longrightarrow$ analyse the process of programming from a semiotic perspective

What is the role of the semiotic modes (assertion, question, query)?

$\longrightarrow$ analyse the role of modes in different programming paradigms