Advanced Formal Object Modeling Techniques

- Polymorphism: Class Union
- Secondary Attributes
- Object Containment

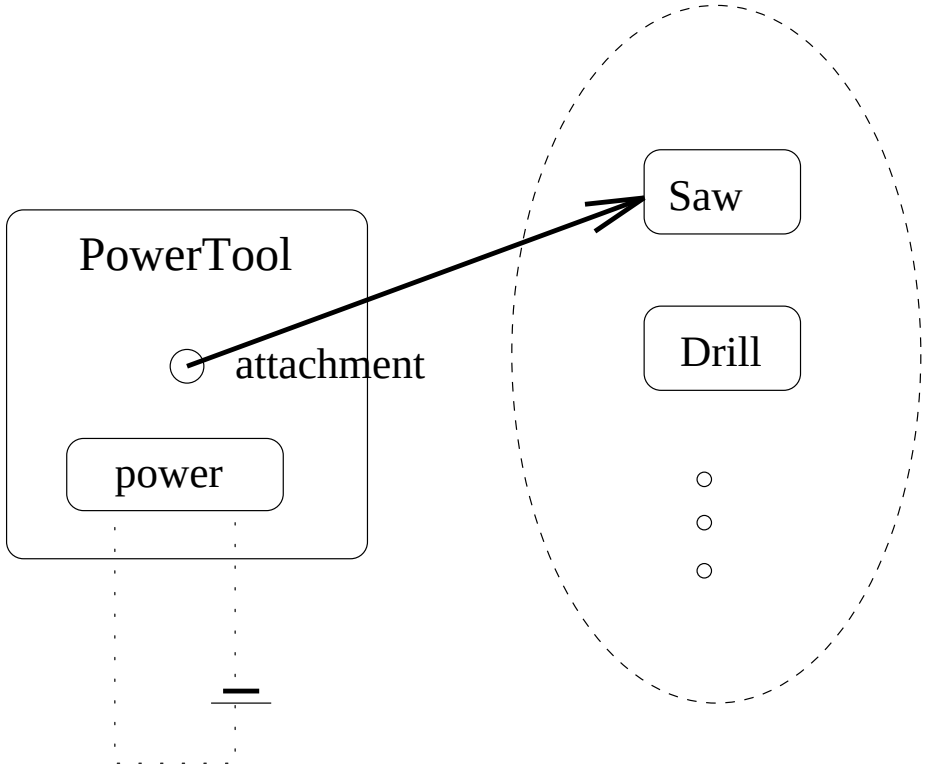
Polymorphism in Specification

- Polymorphic Object Reference:
  if the class of an object reference is not uniquely determined.
- Polymorphic Operation:
  if an operation is defined for each object of a collection and has the same signature.
- Polymorphism and Inheritance Hierarchies

  $$a : \downarrow A$$

Examples of Polymorphism

Example: A Power Tool



Drill

$DrillState ::= still \mid drilling$

$mode : DrillState$

DrillHole

$\Delta(mode)$

$mode' = drilling$

Stop

$\Delta(mode)$

$mode' = still$

Saw

$SawState ::= still \mid sawing$

$mode : SawState$

Cut

$\Delta(mode)$

$mode' = sawing$

Stop

$\Delta(mode)$

$mode' = still$

PowerTool

$att : Drill \textbf{ or } Saw$
$power\_on : \mathbb{B}$

SwitchOn
$\Delta(power\_on)$
...

SwitchOff
$\Delta(power\_on)$
...

Change
$\Delta(att)$
...

$DrillHole \mathrel{\widehat=} [att \in Drill] \land SwitchOn \land att.DrillHole$

$Cut \mathrel{\widehat=} [att \in Saw] \land SwitchOn \land att.Cut$

$Stop \mathrel{\widehat=} SwitchOff \land att.Stop$

Example: Ginger Meggs

GingerMeggs

$pocket : \mathbb{P}(Marble \textbf{ or } String)$

Collect

$\Delta(pocket)$

$x? : Marble \textbf{ or } String$

$x? \notin pocket$

$pocket' = pocket \cup \{x?\}$

Discard

$\Delta(pocket)$

$x! : Marble \textbf{ or } String$

$x! \in pocket$

$pocket' = pocket - \{x!\}$

SelectMarble

$m? : Marble$

$m? \in pocket$

SelectString

$s? : String$

$s? \in pocket$

$PolishMarble \mathrel{\widehat=} SelectMarble \bullet m?.Polish$

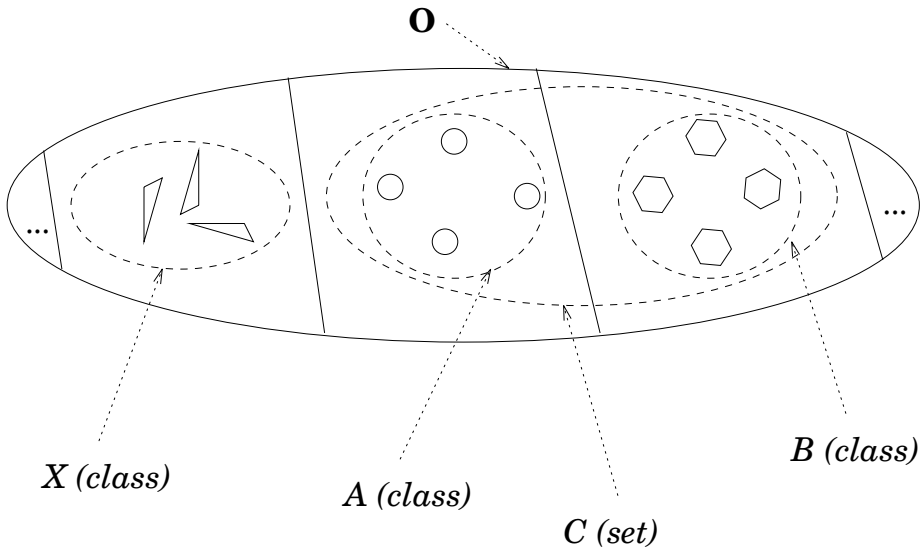
$KnotString \mathrel{\widehat=} SelectString \bullet s?.Knot$

Class Union

- Universe $\mathbb{O}$ of Object Identities
- If A and B are distinct classes in the system,

$$A \subseteq \mathbb{O},\ B \subseteq \mathbb{O} \text{ and } A \cap B = \varnothing.$$

- Class-union $C \mathrel{\widehat{=}} A \cup B$



Polymorphic Core and Extendibility

- Polymorphic Core
- Extending Class Unions
  - Rule:
    a class-union type can be extended, without affecting existing specifications using that type, provided the extended type has the same polymorphic core as the original.

Examples Revisited

Power Tool

$Attachment \mathrel{\widehat=} Drill \cup Saw$

PowerTool

$att : Attachment$

...

...

$Stop \mathrel{\widehat=} SwitchOff \land att.Stop$

Ginger Meggs

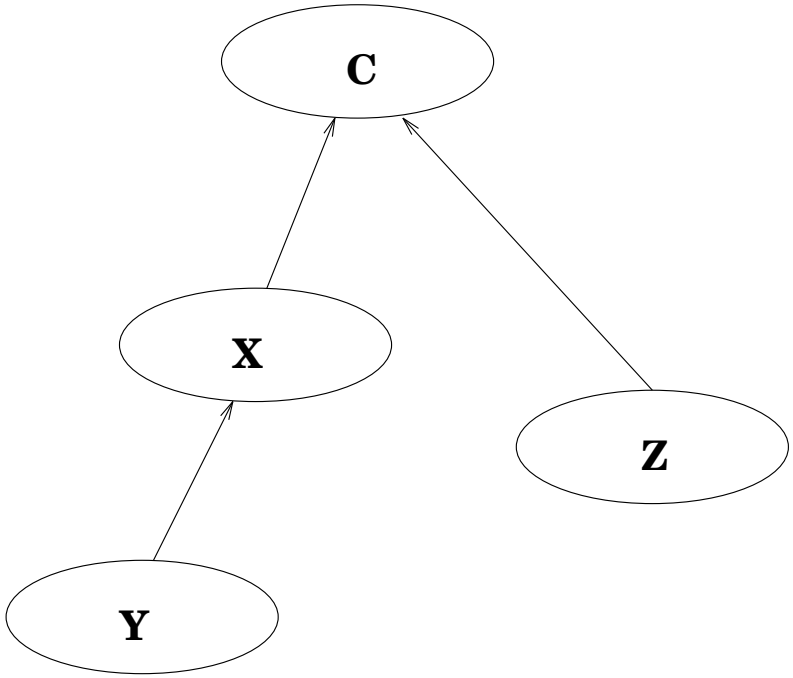
$Treasure \mathrel{\widehat=} Marble \cup String$

GingerMeggs

$pocket : \mathbb{P}\ Treasure$

...

Class Union and Inheritance Hierarchies



$c : \downarrow C$

$c : C \cup X \cup Y \cup Z$

$d : C \cup X \cup Y$

Before Free Types (more standard Z functions)

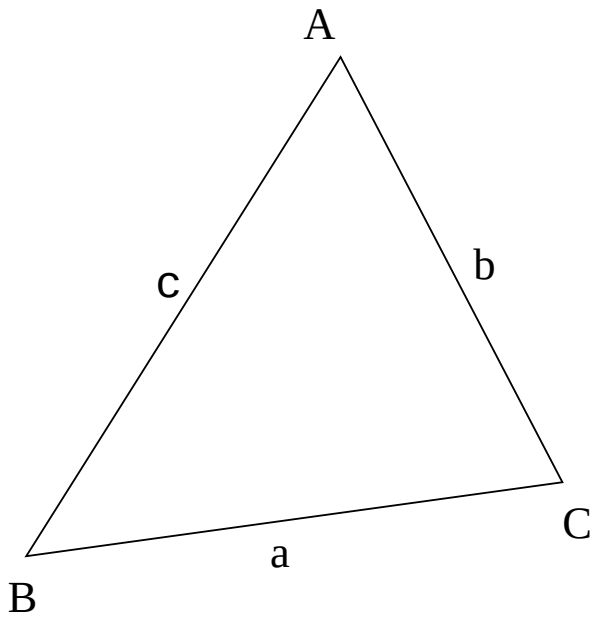
$$
\begin{array}{|l}
\hline
[I, X] \\
\text{disjoint}\_ : \mathbb{P}(I \pfun \mathbb{P}\, X) \\
\_\,\text{partitions}\,\_ : (I \pfun \mathbb{P}\, X) \leftrightarrow \mathbb{P}\, X \\
\hline
\forall\, S : I \pfun \mathbb{P}\, X;\ T : \mathbb{P}\, X \bullet \\
\quad (\text{disjoint}\, S \Leftrightarrow \\
\qquad (\forall\, i, j : \text{dom}\, S \mid i \neq j \bullet S(i) \cap S(j) = \varnothing)) \land \\
\quad (S\, \text{partitions}\, T \Leftrightarrow \\
\qquad \text{disjoint}\, S \land \bigcup \{\ i : \text{dom}\, S \bullet S(i)\ \} = T) \\
\hline
\end{array}
$$

An indexed family of sets S is *disjoint* if and only if each pair of sets $S(i)$ and $S(j)$ for $i \neq j$ have empty intersection. The family S *partitions* a set T if, in addition, the union of all the sets $S(i)$ is T.

Conclusions on Class Union

- Polymorphism restricted to inheritance hierarchies is cumbersome
- Class-union:
  - Widely applicable, Flexible, Extensible
  - Complement to the polymorphic inheritance hierarchies
- Further Reading:
  - J.S. Dong. Living with Free Type and Class Union. *APSEC'95*, IEEE Press, Dec 1995.
  - J.S. Dong & R. Duke. Class Union and Polymorphism. *TOOLS12*, Prentice-Hall, Nov 1993.

Secondary Attributes



$Triangle_{SSS}$

$a, b, c : \mathbb{R}^+$

$Triangle_{SAS}$

$a, B, c : \mathbb{R}^+$

$Triangle_{ASA}$

$A, b, C : \mathbb{R}^+$

$Triangle$

$a, b, c, A, B, C, area, perimeter : \mathbb{R}^+$

many dependencies, e.g. $perimeter = a + b + c$ and $\frac{a}{sin(A)} = \frac{b}{sin(B)} = \frac{c}{sin(C)}$.

Background, Goal and Outline

- Background:
  - Rumbaugh
  - Duke & Rose
- Goal:
  - to investigate the roles of applying secondary attributes in formal object modelling
  - to exploit the implications of secondary attributes
- Outline:
  - Basic Usage (Hotel Management System)
  - Adding Clarity (Complex number variable)
  - Environmental Dependency: Object Sharing (Game Sharing)
  - Dependency Chain and Recursion (Simple Predicates)

Basic Usage (Hotel Management System)

HotelManagementSystem

$$
\begin{array}{l}
rooms : \mathbb{P}\, Room \\
occupied : \mathbb{P}\, Room \\
\Delta \\
vacant : \mathbb{P}\, Room \\
\hline
occupied \subseteq rooms \\
vacant = rooms - occupied
\end{array}
$$

CheckIn

$$
\begin{array}{l}
\Delta(occupied) \\
r? : Room \\
\hline
r? \in vacant \\
occupied' = occupied \cup \{r?\}
\end{array}
$$

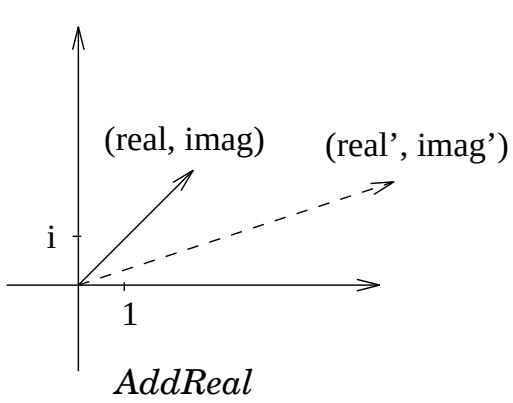
CheckOut

$$
\begin{array}{l}
\Delta(occupied) \\
r? : Room \\
\hline
r? \in occupied \\
occupied' = occupied - \{r?\}
\end{array}
$$

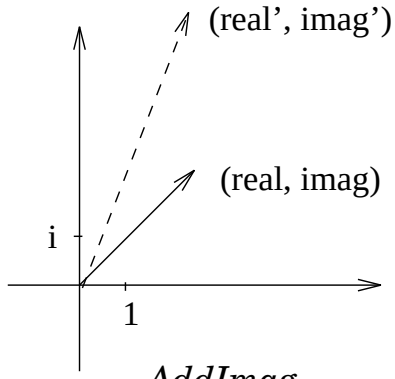
ListVacancy

$$
\begin{array}{l}
rooms! : \mathbb{P}\, Room \\
\hline
rooms! = vacant
\end{array}
$$

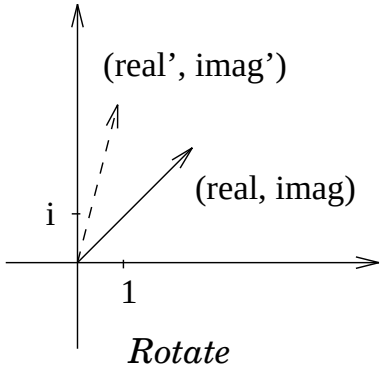
Clarity in Specification (Complex number variable)



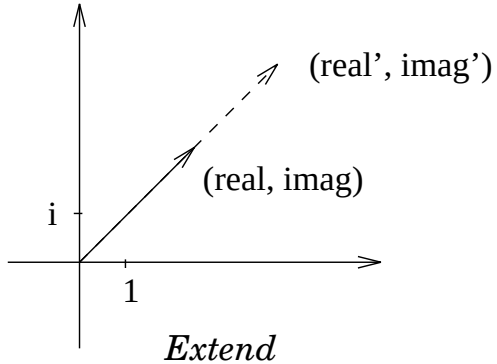
AddReal



AddImag



Rotate



Extend

ComplexNumVar

$$real, imag : \mathbb{R}$$
$$\Delta$$
$$modulus, argument : \mathbb{R}$$

$$modulus = \sqrt{real^2 + imag^2} \land sin(argument) \times modulus = imag$$
$$cos(argument) \times modulus = real$$

AddReal

$$\Delta(real)$$
$$r? : \mathbb{R}$$

$$real' = real + r?$$

AddImag

$$\Delta(imag)$$
$$i? : \mathbb{R}$$

$$imag' = imag + i?$$

Rotate

$$\Delta(real, imag)$$
$$\alpha? : \mathbb{R}$$

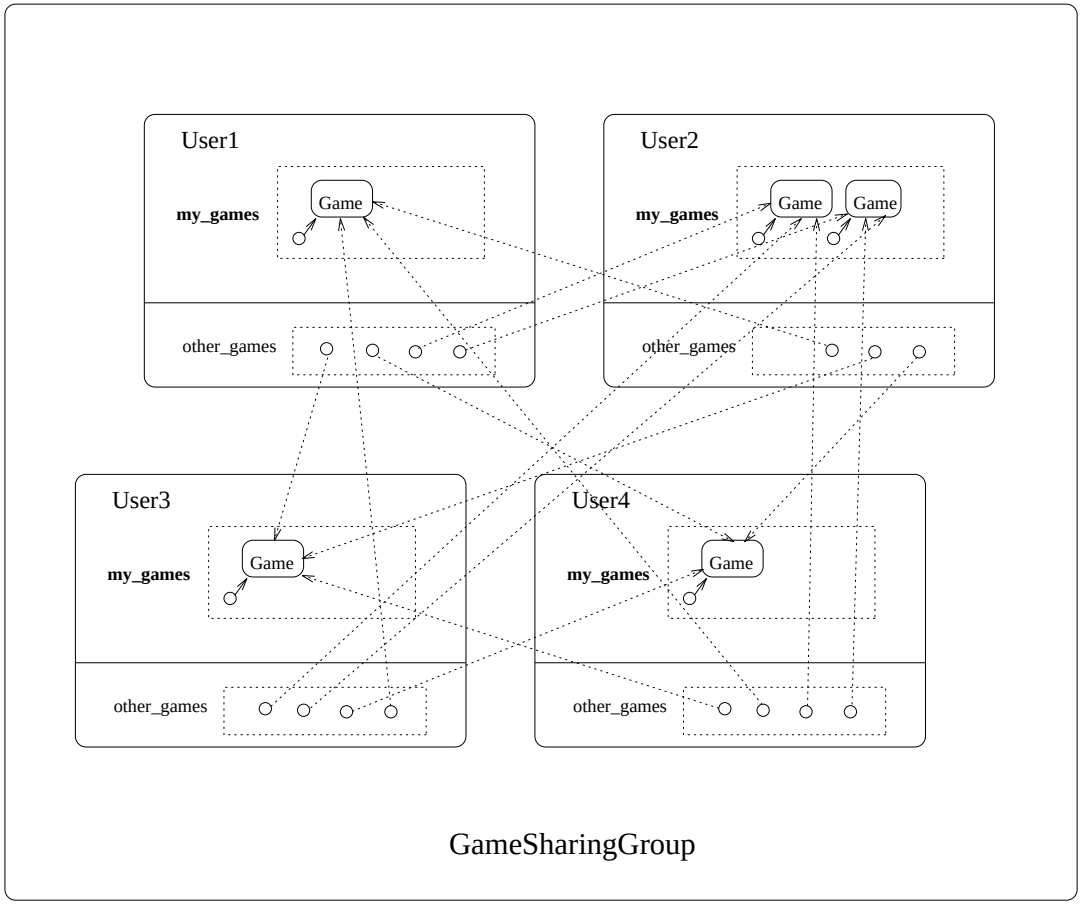
$$argument' = argument + \alpha?$$
$$modulus' = modulus$$

Extend

$$\Delta(real, imag)$$
$$l? : \mathbb{R}^+$$

$$modulus' = modulus + l?$$
$$argument' = argument$$

Dependency on Environment: Object Sharing



Game

...

Play

...

User

$my\_games : \mathbb{P}\, Game_{\copyright}$

Δ

$other\_games : \mathbb{P}\, Game$

AddGame

$\Delta(my\_games)$

$g? : Game$

$g? \notin my\_games$

$my\_games' = my\_games \cup \{g?\}$

DelGame

$\Delta(my\_games)$

$g? : Game$

$g? \in my\_games$

$my\_games' = my\_games - \{g?\}$

Choose

$g? : Game$

$g? \in (my\_games \cup other\_games)$

$Play \mathrel{\widehat=} Choose \bullet g?.Play$

GameSharingGroup

$$users : \mathbb{P}\ Users$$

$$\forall\, u_1 : users \bullet u_1.other\_games = \bigcup \{u_2 : users \mid u_2 \neq u_1 \bullet u_2.my\_games\}$$

Select

$$u? : User$$

$$u? \in users$$

SelectTwo

$$u_1?, u_2? : User$$

$$u_1? \neq u_2?$$
$$\{u_1?, u_2?\} \subseteq users$$

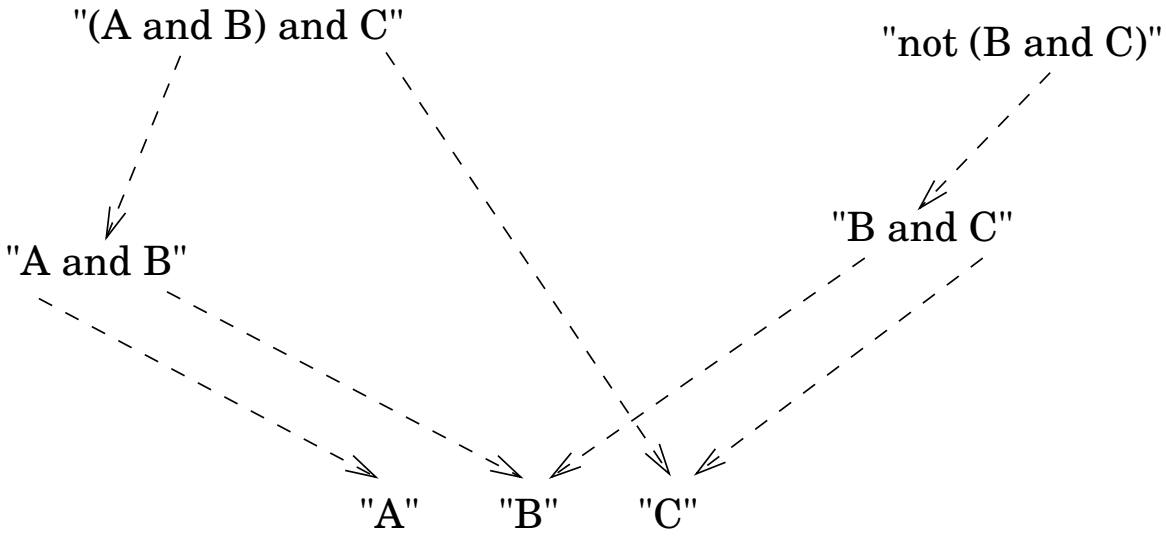
$$Play \mathrel{\widehat=} Select \bullet u?.Play$$

$$DiscardGame \mathrel{\widehat=} Select \bullet u?.DelGame$$

$$NewGame \mathrel{\widehat=} Select \bullet u?.AddGame$$

$$Transfer \mathrel{\widehat=} SelectTwo \bullet u_1?.DelGame \land u_2?.AddGame$$

Dependency Chain and Recursion



$$Predicate \mathrel{\widehat=} LogicVar \cup Conjunction \cup Negation$$

BasePredicate

$$\Delta$$
$$value : \mathbb{B}$$
$$subs : \mathbb{P}\, Predicate$$

LogicVar

BasePredicate

$$name : Id$$
$$content : \mathbb{B}$$

$$value = content$$
$$subs = \varnothing$$

Change

$$\Delta(content)$$
$$value? : \mathbb{B}$$

$$content' = value?$$

Conjunction

BasePredicate

$$l, r : Predicate$$

$$value = l.value \land r.value$$
$$subs = l.subs \cup r.subs \cup \{l, r\}$$
$$self \notin subs$$

Negation

BasePredicate

$$p : Predicate$$

$$value = \neg\, p.value$$
$$subs = p.subs \cup \{p\}$$
$$self \notin subs$$

PredStore

$$preds : \mathbb{P}\, Predicates$$
$$\Delta$$
$$vars : \mathbb{P}\, LogicVar$$

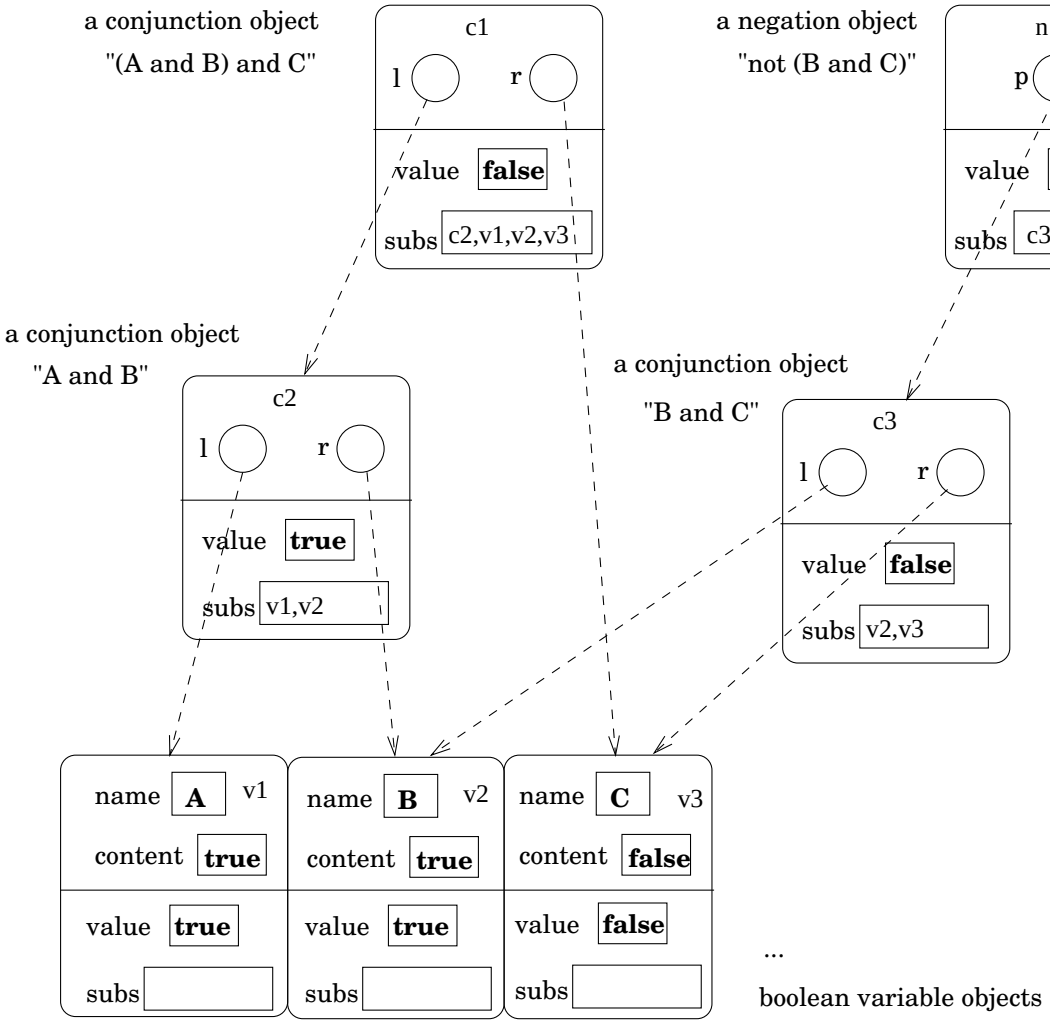
$$vars = (\bigcup \{p : preds \bullet p.subs\} \cup preds) \cap LogicVar$$
$$\#\{v : vars \bullet v.name\} = \#vars$$

Output

$$pred? : Predicate$$
$$value! : \mathbb{B}$$

$$pred? \in preds \land value! = pred?.value$$

$$Change \mathrel{\widehat=} [var? : vars] \bullet var?.Change$$


a conjunction object
"(A and B) and C"
c1
l
r
value
false
subs
c2,v1,v2,v3
a negation object
"not (B and C)"
n1
p
value
true
subs
c3,v2,v3
a conjunction object
"A and B"
c2
l
r
value
true
subs
v1,v2
a conjunction object
"B and C"
c3
l
r
value
false
subs
v2,v3
name
A
v1
content
true
value
true
subs
name
B
v2
content
true
value
true
subs
name
C
v3
content
false
value
false
subs
...
boolean variable objects

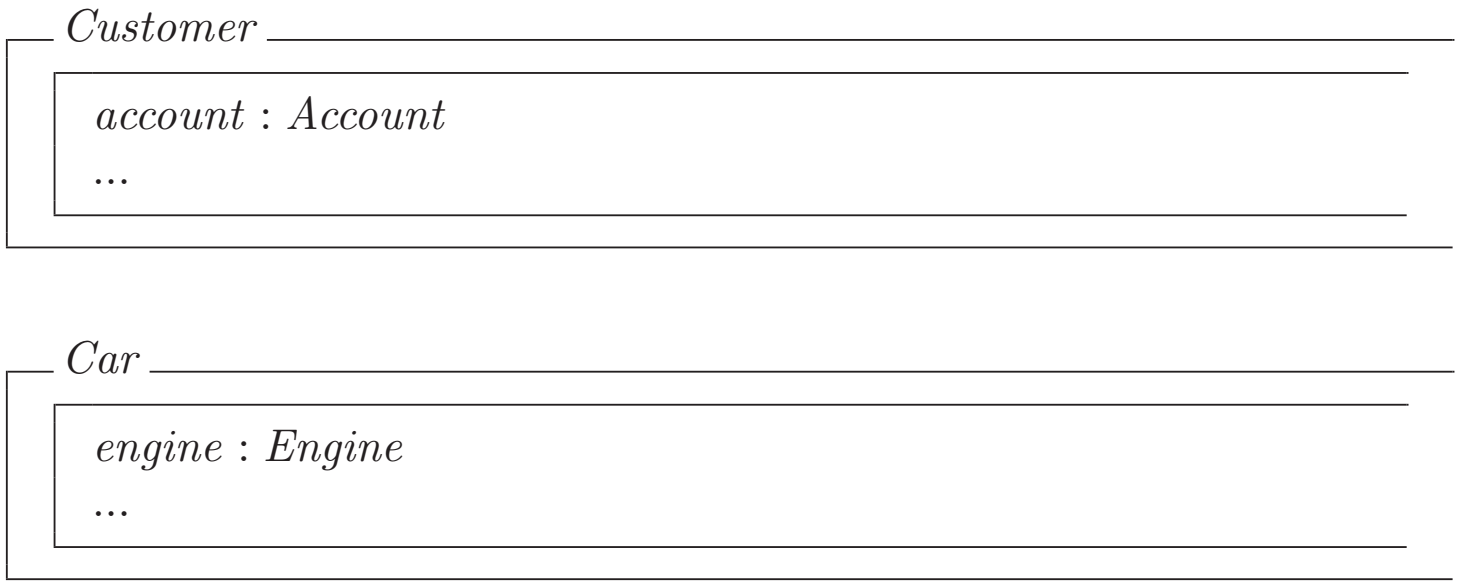
Conclusion on Secondary Attributes

Small case studies analyse and demonstrate the three different usages of secondary attributes in formal modelling, in summary:

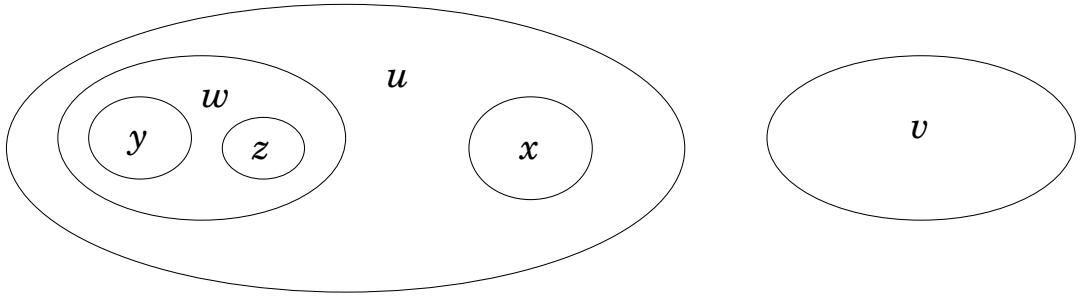
- improving the clarity and simplicity of object-oriented system specifications in general;
- capturing a notion of object sharing; and
- constructing recursive definitions.
- Further Reading:
  - J.S. Dong, G. Rose and R. Duke. The Role of Secondary Attributes in Formal Object Modelling. In *the IEEE International Conference on Engineering Complex Computer Systems (ICECCS'95)*. IEEE Press, Nov 1995.

Object Containment

A Difference:



Properties of Containment



(1) an object cannot be directly contained within two distinct objects; and

(2) an object cannot directly or indirectly contain itself.

$$dcon : \mathbb{O} \leftrightarrow \mathbb{O}$$

$$dcon^{\sim} \in \mathbb{O} \nrightarrow \mathbb{O}$$
$$\nexists ob : \mathbb{O} \bullet ob \ \underline{dcon^{+}} \ ob$$

Example: Russian Dolls

$$Doll \mathrel{\widehat=} Solid \cup Hollow$$

Solid

$$dolls : \mathbb{P}\, Doll$$

$$dolls = \varnothing$$

Hollow

$$inside : Doll$$
$$dolls : \mathbb{P}\, Doll$$

$$dolls = \{inside\} \cup inside.dolls$$
$$self \notin dolls$$

Example: Terminal Location

Terminal

$\ldots$

Room

$ts : \mathbb{P}\ Terminal$

Building

$rs : \mathbb{P}\ Room$

$\forall\ r_1, r_2 : rs \bullet$
$\quad r_1 \neq r_2 \Rightarrow r_1.ts \cap r_2.ts = \varnothing$

Campus

$bs : \mathbb{P}\ Building$

$\forall\ b_1, b_2 : bs \bullet b_1 \neq b_2 \Rightarrow$
$\quad b_1.rs \cap b_2.rs = \varnothing$
$\quad \forall\ r_1 : b_1.rs;\ r_2 : b_2.rs \bullet r_1.ts \cap r_2.ts = \varnothing$

Capturing the Geometry of Containment

Suppose each class has an implicitly declared attribute

$$dcontain : \mathbb{P}\,\mathbb{O}$$

where the value of *dcontain* is the set of directly contained objects. The properties of the relation *dcon* (as stated earlier) imply invariant conditions on the system that need not be stated explicitly. In terms of the attribute *dcontain* these conditions are:

$$\forall\, ob_1, ob_2 : \mathbb{O} \bullet \\ ob_1 \neq ob_2 \Rightarrow ob_1.dcontain \cap ob_2.dcontain = \varnothing$$

(i.e. no object is directly contained in two distinct objects)

$$\nexists s : \mathrm{seq}\,\mathbb{O} \bullet \\ \#s > 1 \\ \forall\, i : 1 \,..\, \#s - 1 \bullet s(i+1) \in s(i).dcontain \\ s(1) = s(\#s)$$

(i.e. no object directly or indirectly contains itself).

Russian Dolls

Solid

$$dcontain = \varnothing$$

Hollow

$$inside : Doll$$

$$dcontain = \{inside\}$$

Terminal Location

Room

$$ts : \mathbb{P}\, Terminal$$

$$dcontain = ts$$

Building

$$rs : \mathbb{P}\, Room$$

$$dcontain = rs$$

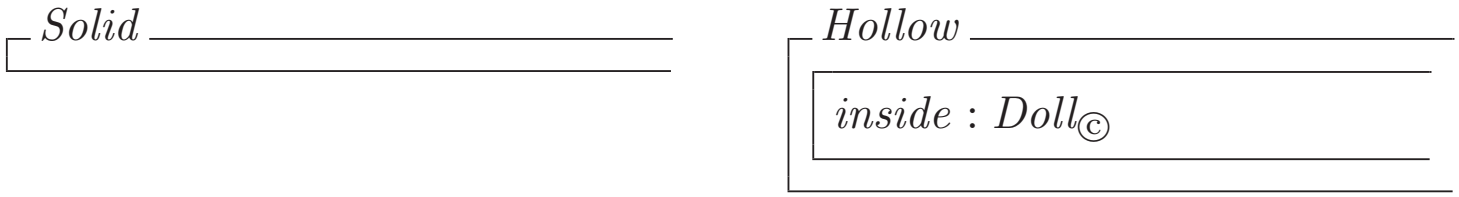
Campus

$$bs : \mathbb{P}\, Building$$

$$dcontain = bs$$

A Notational Simplification

Russian Dolls



Terminal Location

Object Migration

Room

$ts : \mathbb{P}\, Terminal_{©}$

Add

$\Delta(ts)$
$t? : Terminal$

$t? \notin ts \land ts' = ts \cup \{t?\}$

Remove

$\Delta(ts)$
$t? : Terminal$

$t? \in ts \land ts' = ts - \{t?\}$

Building

$rs : \mathbb{P}\, Room_{©}$

SelectRoom

$r? : Room$

$r? \in rs$

Select2Rooms

$r_1?, r_2? : Room$

$r_1? \neq r_2? \land \{r_1?, r_2?\} \subseteq rs$

$Add \mathrel{\widehat{=}} SelectRoom \bullet r?.Add$

$Remove \mathrel{\widehat{=}} SelectRoom \bullet r?.Remove$

$Transfer \mathrel{\widehat{=}} Select2Rooms \bullet$
$r_1?.Remove \parallel r_2?.Add$

Containment in Programming Language

Eiffel:

class Car **feature**

$w1, w2, w3, w4$:**expanded class** Wheel

owner: Person

...

C++:

class Car {

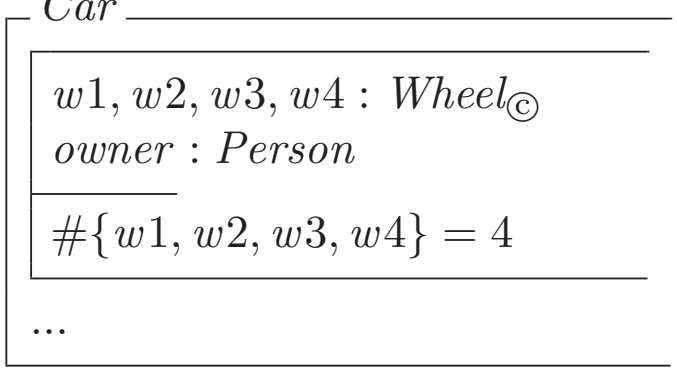
Wheel w1,w2,w3,w4

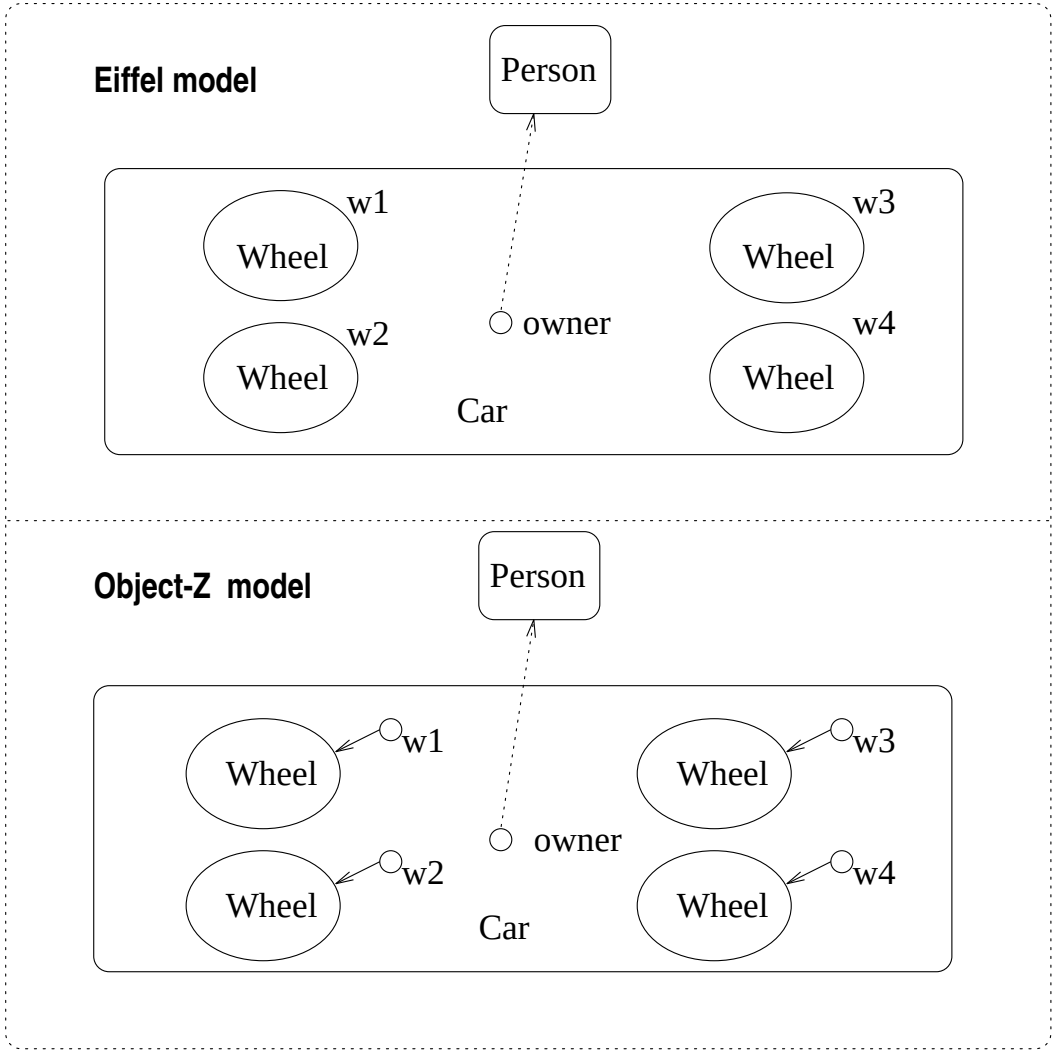
Person *owner

...

}

Object-Z:




Eiffel model
Person
w1
Wheel
w3
Wheel
owner
w2
Wheel
w4
Wheel
Car
Object-Z model
Person
Wheel
w1
Wheel
w3
owner
Wheel
w2
Wheel
w4
Car

Containment and Control

Account

$$bal, lim : \mathbb{Z}; \quad owners : \mathbb{P}\, Customer$$

$$bal \geq lim \land owner \neq \varnothing$$

Deposit

$$\Delta(bal); \quad n? : \mathbb{N}$$

$$bal' = bal + n?$$

Withdraw

$$\Delta(bal); \quad n? : \mathbb{N}$$

$$bal' = bal - n?$$

ChangeLimit

$$\Delta(lim); \quad n? : \mathbb{N}$$

$$lim' = n?$$

Bank

$$accs : \mathbb{P}\, Account_{\copyright}$$

$$ChangeLimit \mathrel{\widehat=} [a : accs] \bullet a.ChangeLimit$$

Customer

$$accs : \mathbb{P}\, Account$$

$$\forall\, a : accs \bullet self \in a.owners$$

$$Deposit \mathrel{\widehat=} [a : accs] \bullet a.Deposit$$

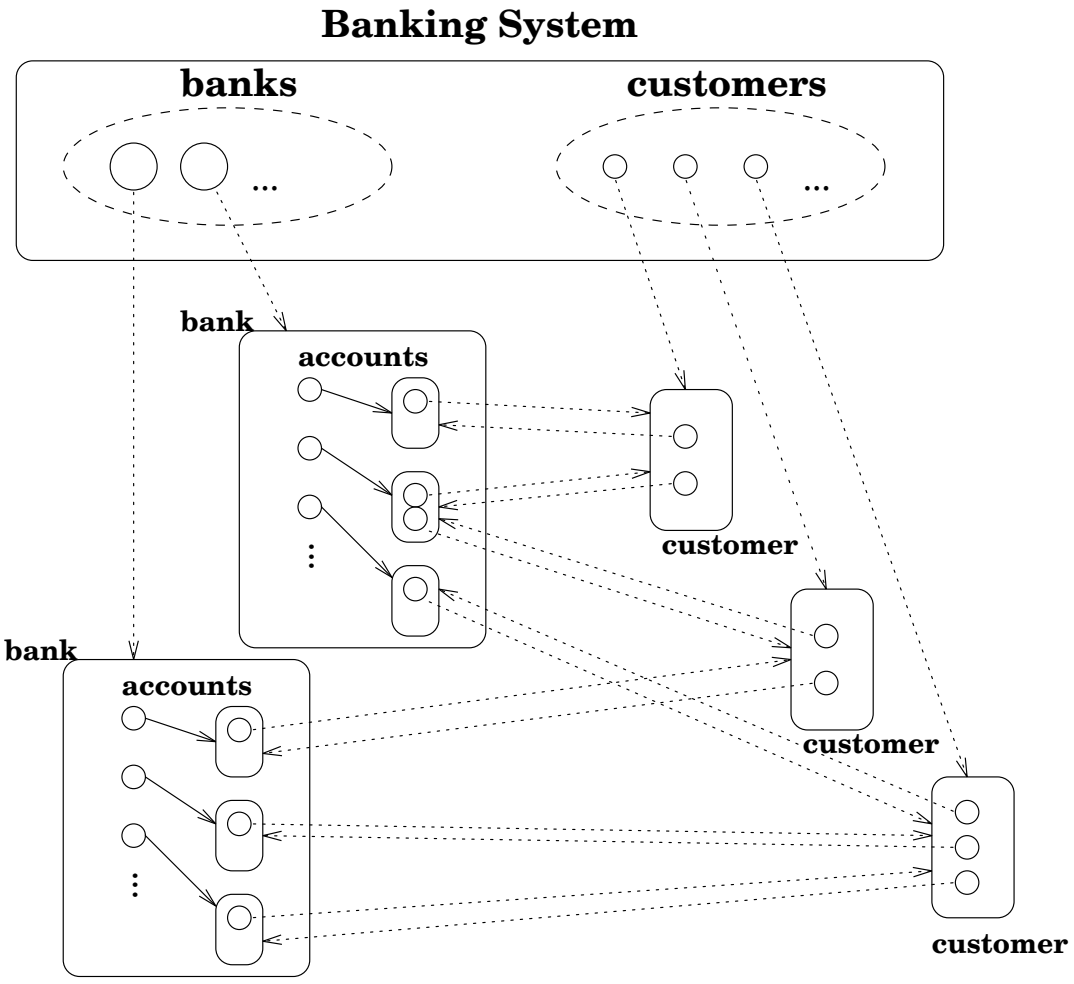
$$Withdraw \mathrel{\widehat=} [a : accs] \bullet a.Withdraw$$

BankingSystem

$$banks : \mathbb{P}\, Bank$$
$$customers : \mathbb{P}\, Customer$$

$$\bigcup\{b : banks \bullet b.accs\} = \bigcup\{c : customers \bullet c.accs\}$$

$$Deposit \mathrel{\widehat=} [c : customers] \bullet c.Deposit$$
$$Withdraw \mathrel{\widehat=} [c : customers] \bullet c.Withdraw$$
$$ChangeLimit \mathrel{\widehat=} [b : banks] \bullet b.ChangeLimit$$


Banking System
banks
customers
bank
accounts
customer
bank
accounts
customer
customer

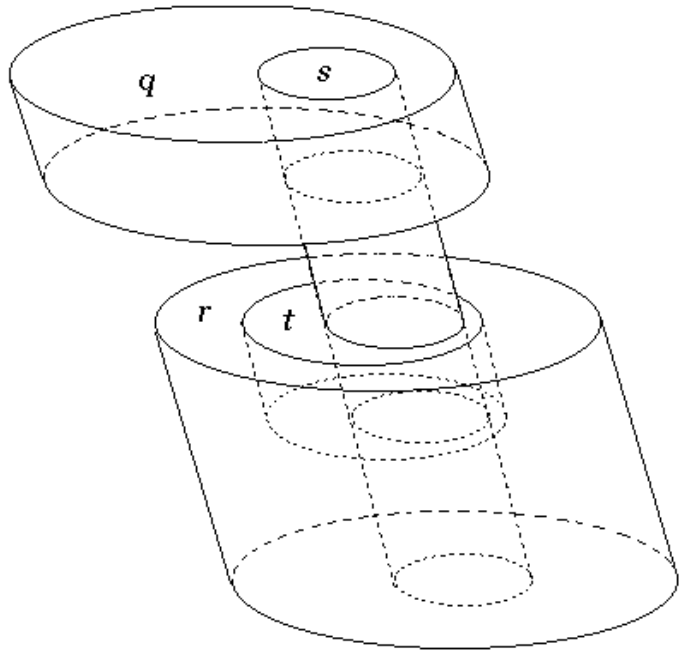
Modelling Sharable Containment

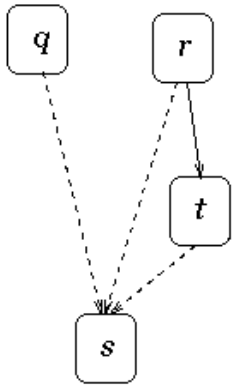
$$
\begin{array}{|l}
\hline
sdcon, dcon, DC : \mathbb{O} \leftrightarrow \mathbb{O} \\
\hline
DC = dcon \cup sdcon \\
dcon^{\sim} \in \mathbb{O} \nrightarrow \mathbb{O} \\
\nexists ob : \mathbb{O} \bullet ob \ \underline{DC^{+}} \ ob \\
dcon \cap sdcon = \varnothing \\
\hline
\end{array}
$$

...

$sdcontain : \mathbb{P}\,\mathbb{O}$

...





Examples:

Walls, Rooms and Buildings

Wall

...

Room

$ws : \mathbb{P}\ Wall_{\textcircled{S}}$

...

Building

$rs : \mathbb{P}\ Room_{\textcircled{C}}$
$ws : \mathbb{P}\ Wall_{\textcircled{C}}$

$\bigcup\{r : rs \bullet r.ws\} \subseteq ws$

...

Buildings, Streets and Suburbs

Building

...

Street

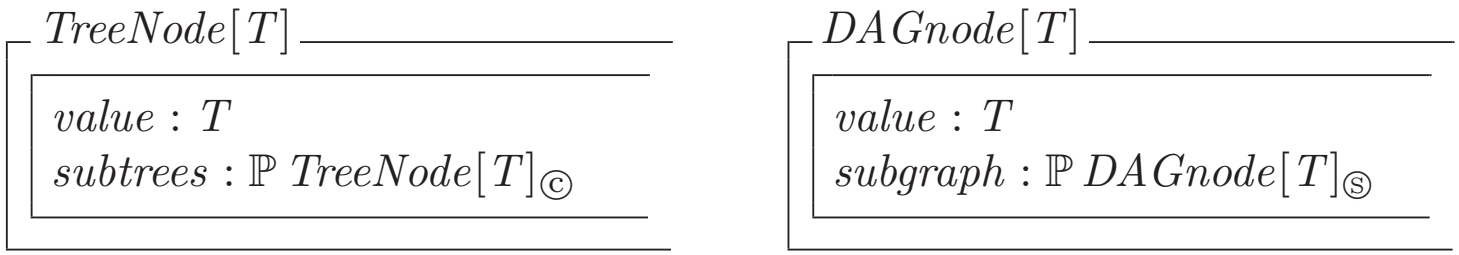
...

Suburb

$bs : \mathbb{P}\ Building_{\textcircled{C}}$
$ss : \mathbb{P}\ Street_{\textcircled{S}}$

...

Containment as an Abstraction



Further Reading

J.S. Dong and R. Duke. The Geometry of Object Containment. *Object-Oriented Systems (OOS)* journal 2(1):41-63, Chapman & Hall, 1995.