Semantics of Edge Visitor Methods

Karl and Pengcheng

Kinds of edge methods

- construction edges
  - cbefore / cafter / caround
- repetition edges
  - rbefore / rafter / raround
- strategy edges
  - sbefore / safter / saround
- In the following focus is on before methods.

Addition to Visitor

has a method

`CEdge getCEdge()`

that returns the CEdge of the

current or most recent construction edge being traversed. The motivation is to fill in the blanks about the edge being traversed. The interface of cbefore may not uniquely determine an edge. DJ now also supports edge patterns.

Example

- void cbefore(Source s);
  - `-> Source, *, *`
- This s an example of a cbefore method on a construction edge that maps several edges. getCEdge() will give us the details of the current edge and the programmer may use conditional statements to make the behavior dependent on the details of the edge.

CEdge

```
CEdge =
  [<sourceName> String]
  [<partName> String]
  [<targetName> String]
  [<edgeKind> String].
// derived / public, protected, private
```

Visitor Methods for Construction Edges

- void cbefore_x(Source s, Target t);
  - `-> Source,x,Target`
- void cbefore(Source s, Target t);
  - `-> Source, *, Target`
- void cbefore_x(Source s);
  - `-> Source, x, *`
- void cbefore(Source s);
  - `-> Source, *, *`

All but the first method needs getCEdge to get edge details.

Visitor Methods for Construction Edges

- void cbefore_x(Target t);
  - `-> *,x,Target`
- void cbefore(Target t);
  - `-> *,*,Target`
- void cbefore_x();
  - `-> *,x,*`
- void cbefore(); // all edges
  - `-> *,*,*`

Programming with Strategies

- Strategies are abstractions of class graphs and it is useful to program directly with those abstractions. Therefore we extend DJ in two ways by allowing:
  - to check whether the traversal is currently in the scope of a subtraversal
    - if (SEdges(sg)) {} (e.g., inside a cbefore method) checks whether the currently active strategy graph edges are contained in the strategy sg. Notice that e.g. during the traversal of an object graph edge, multiple strategy graph edges may be active.
  - edge methods on strategies

Programming with Strategies

- to check whether the traversal is currently in the scope of a subtraversal
  - We want to talk about the state of the traversal in *before methods. This allows us to distinguish how we arrived at an object graph node.
- edge methods on strategies
  - Strategies are built out of simpler strategies. We want to attach before and after code to the simpler strategies.

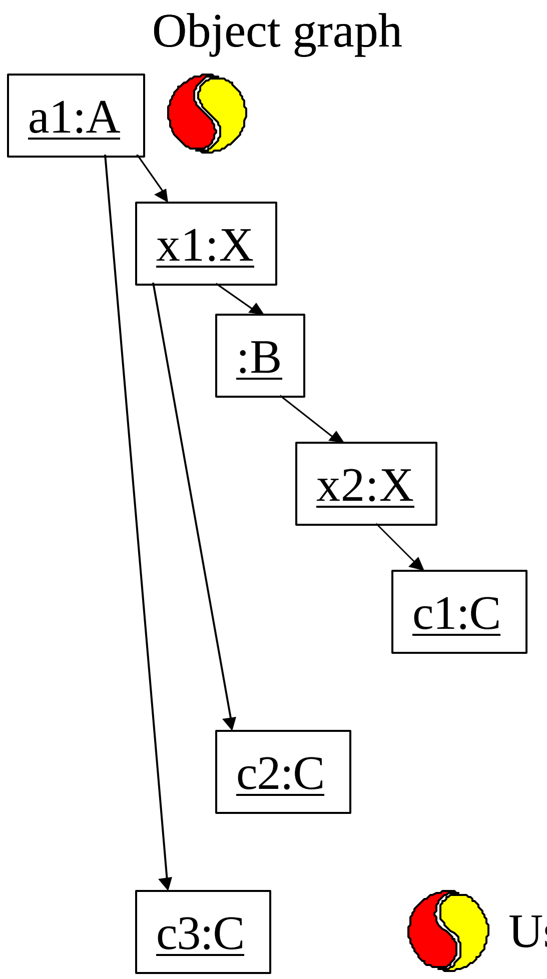
Programming with Strategies

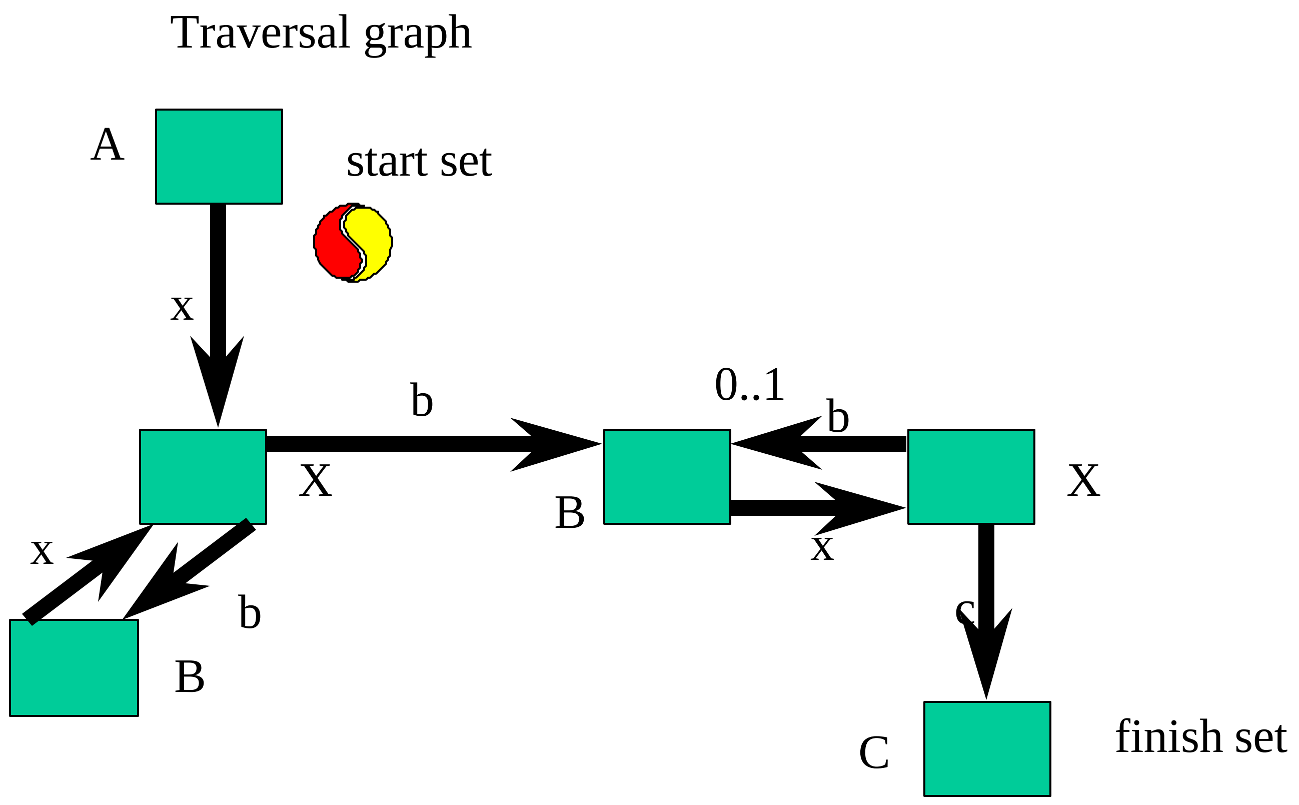
- At a given point during the traversal a set of strategy graph edges is active. I think of it as the strategy edges corresponding to traversal graph edges on which a token travels. But there is a more direct explanation: the set of strategy graph edges that are needed for the expansion into object paths.
- The following example illustrates the concept of active strategy graph edges. Basically when several tokens are distributed in different copies of the class graph inside the traversal graph, several edges are active.

Active strategy graph edges: `A -> B for :A->x1:X`

```
strategy:
{A -> B
 B -> C}
```

Short-cut



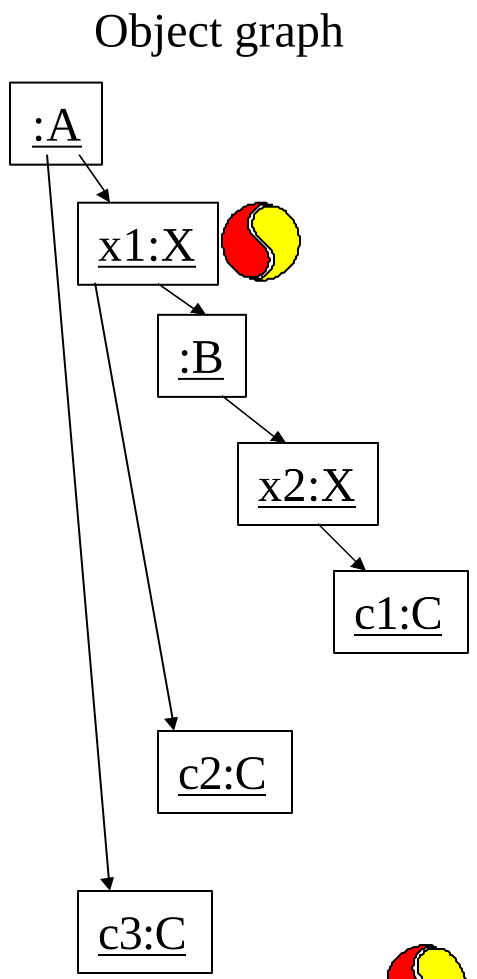


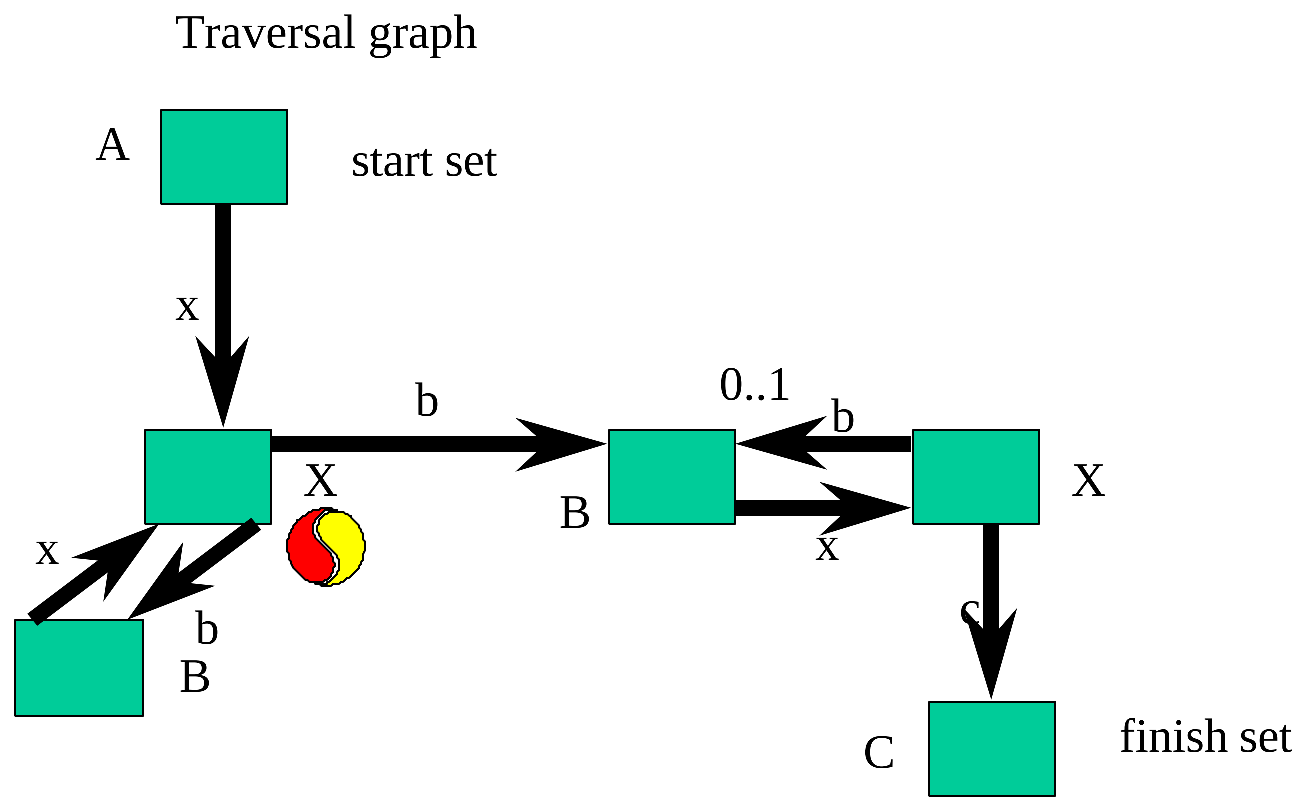
Used for token set and currently active object

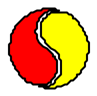
Active strategy graph edges: `A -> B for x1:X->:B`

```
strategy:
{A -> B
 B -> C}
```

Short-cut



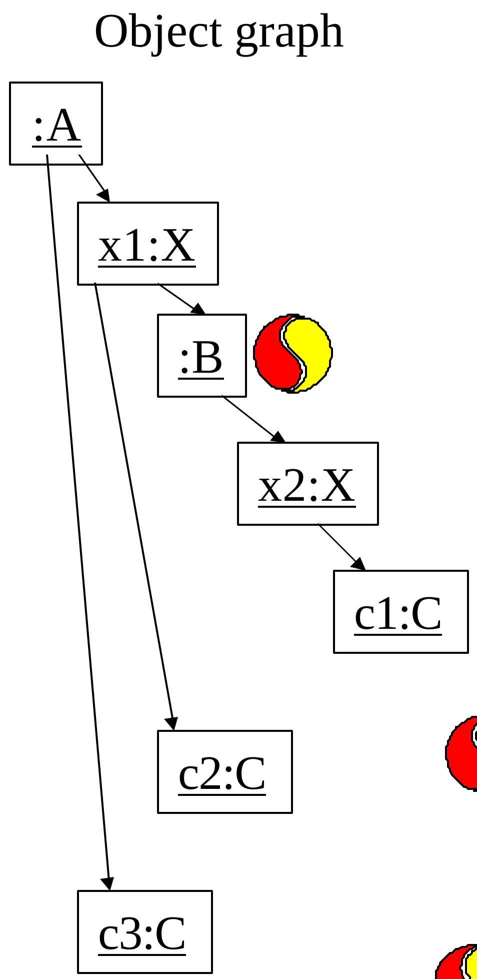


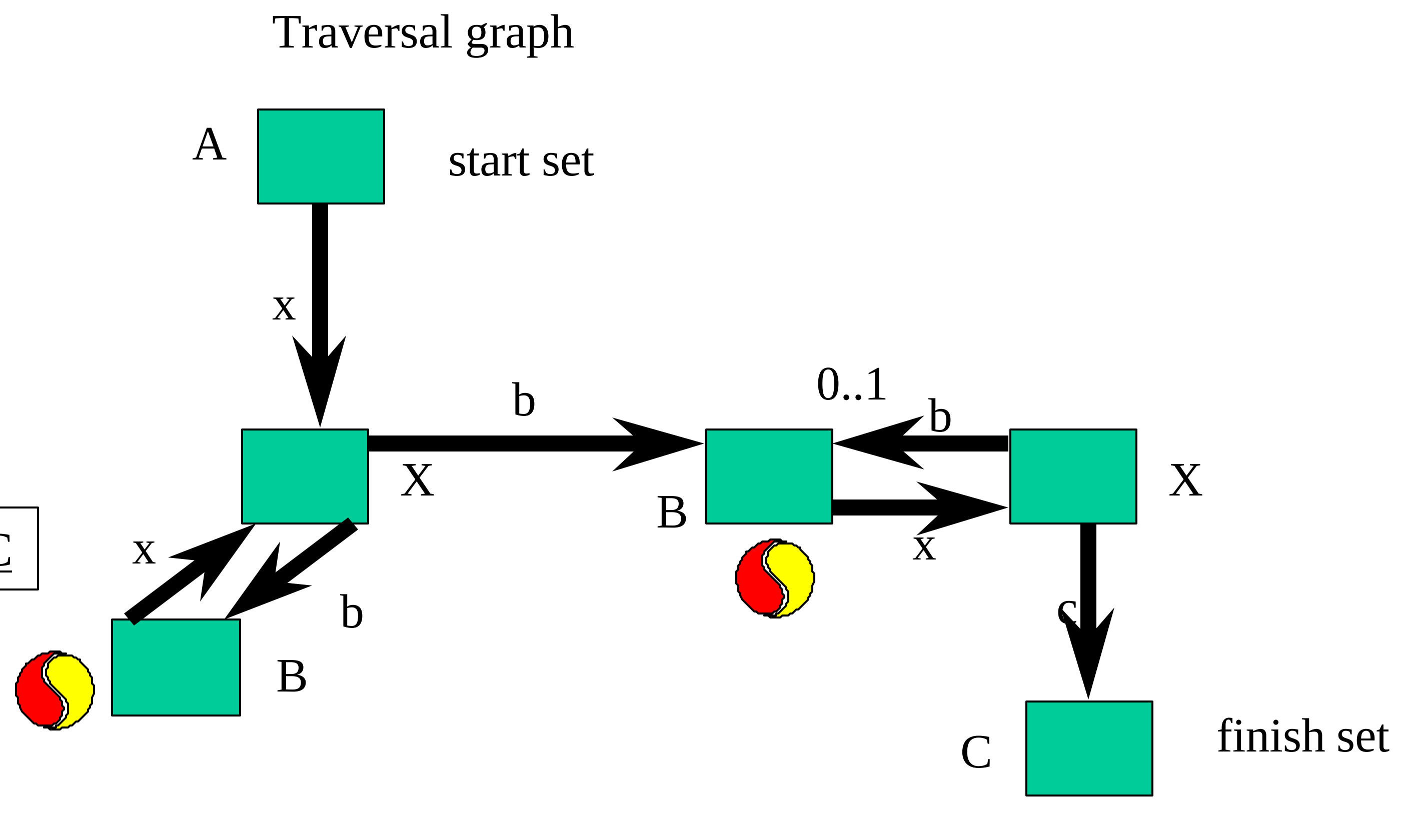

Used for token set and currently active object

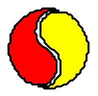
Active strategy graph edges: A->B B->C for :B->x2:X

Short-cut

```
strategy:
{A -> B
 B -> C}
```



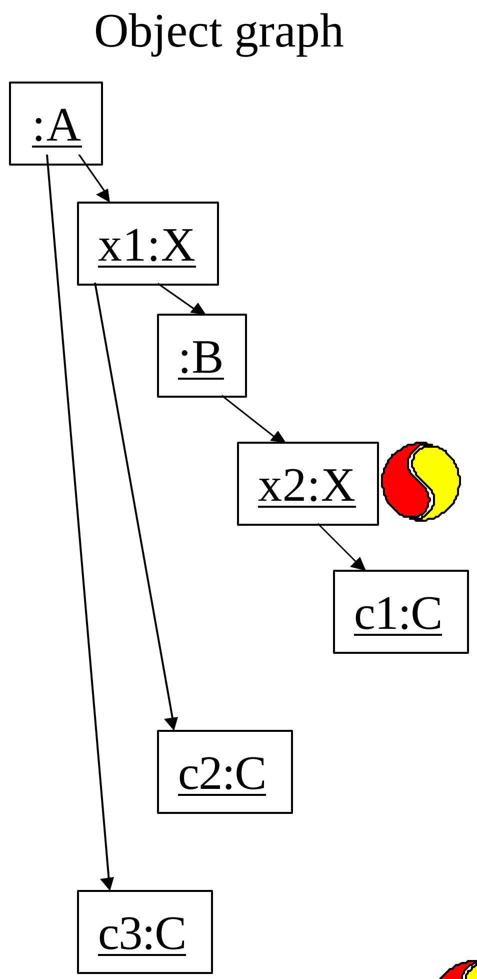


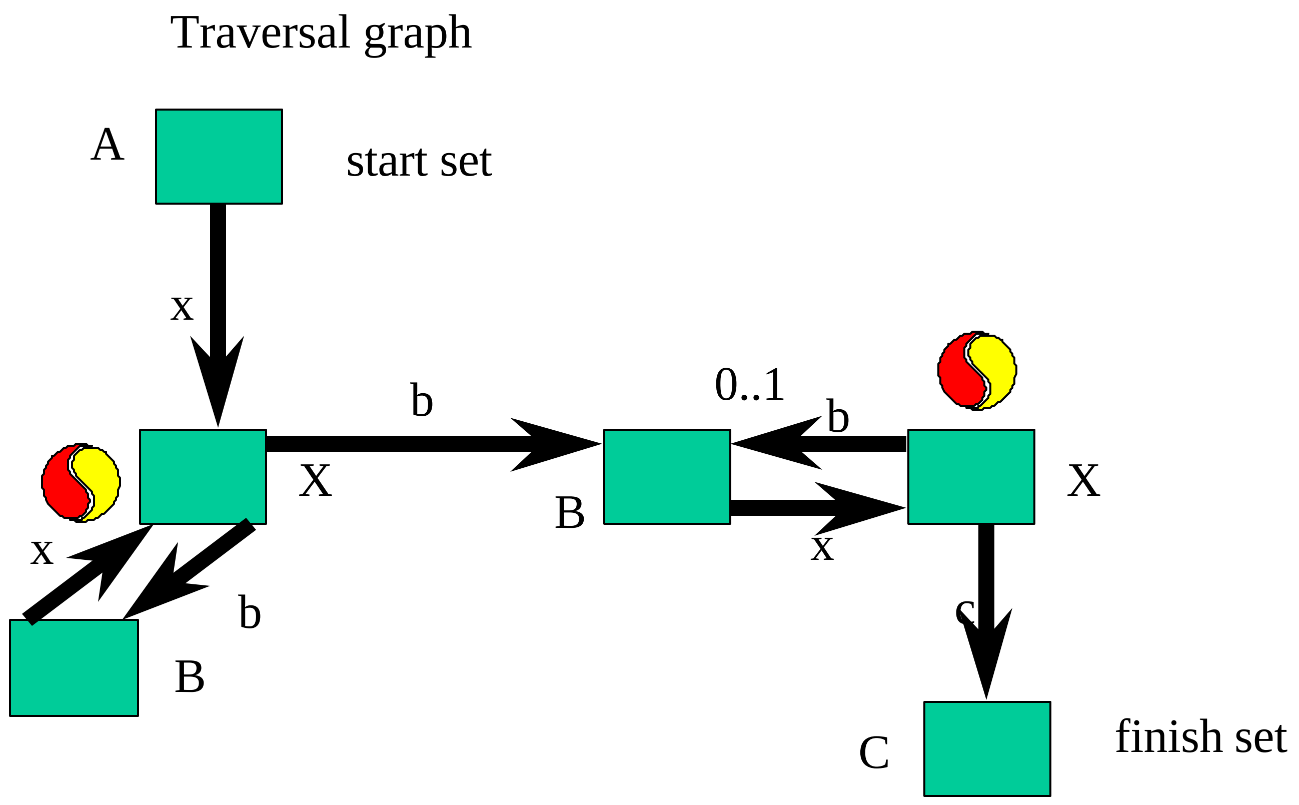
 Used for token set and currently active object

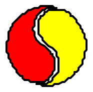
Active strategy graph edges: `B->C for x2:X->c1:C`

Short-cut

```
strategy:
{A -> B
 B -> C}
```

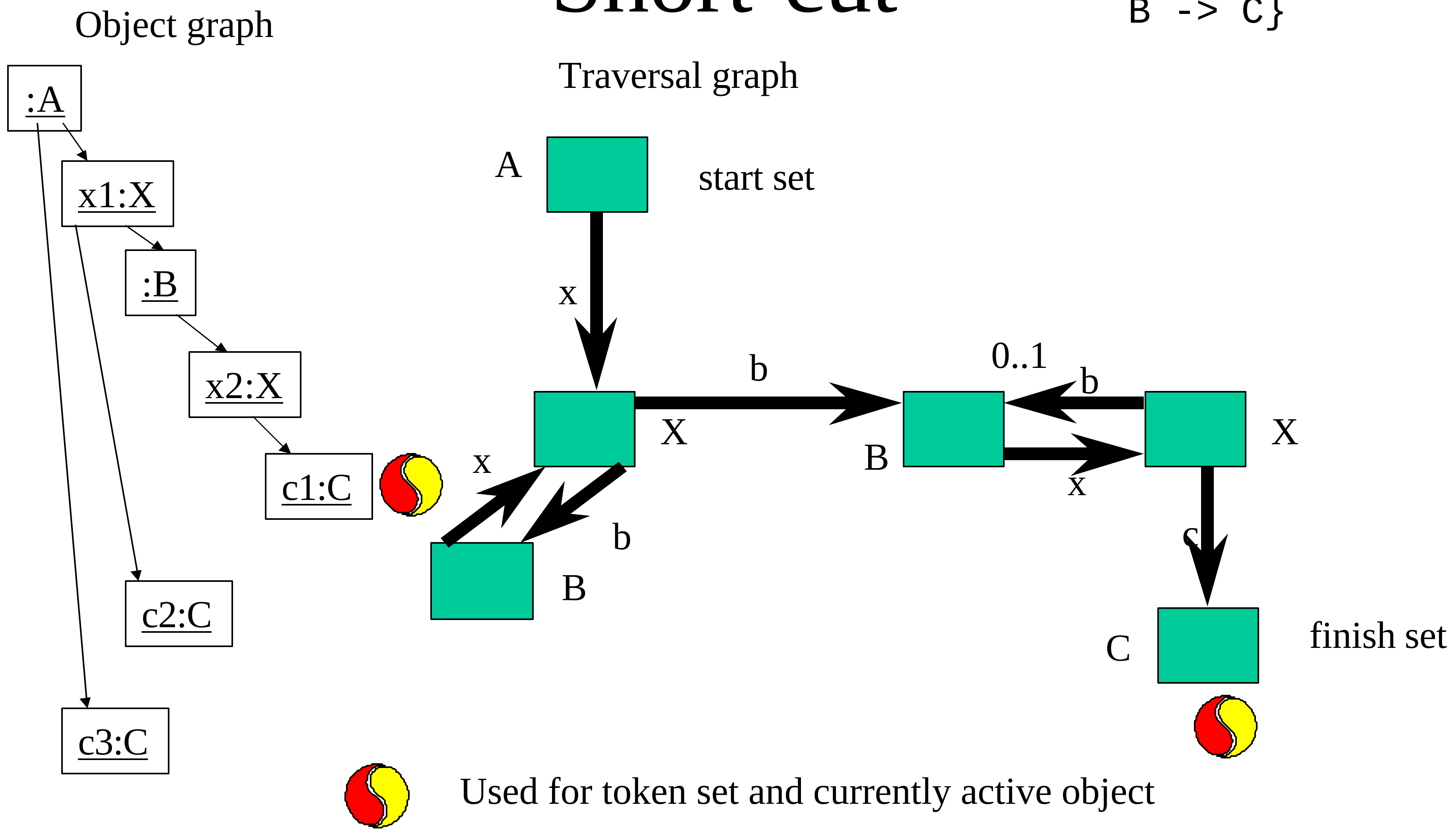





Used for token set and currently active object

Short-cut

```
strategy:
{A -> B
 B -> C}
```



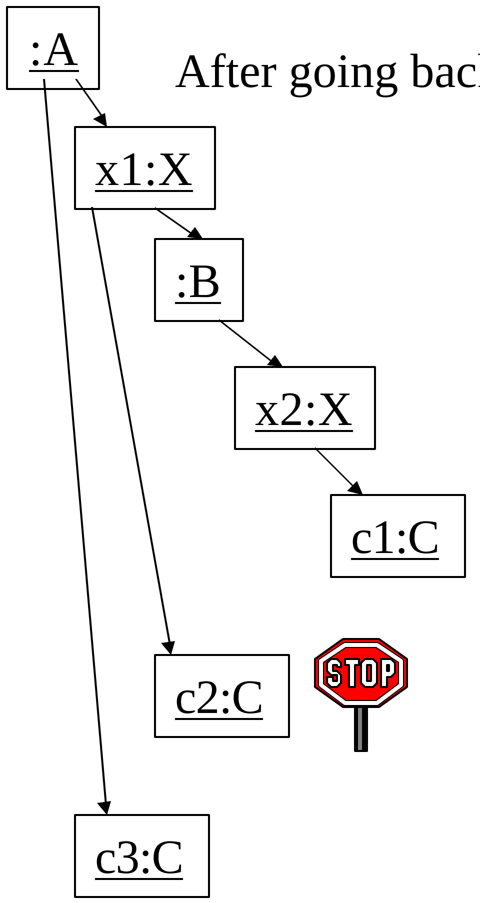
Used for token set and currently active object

Active strategy graph edges: `A->B for :A->x1:X`

Short-cut

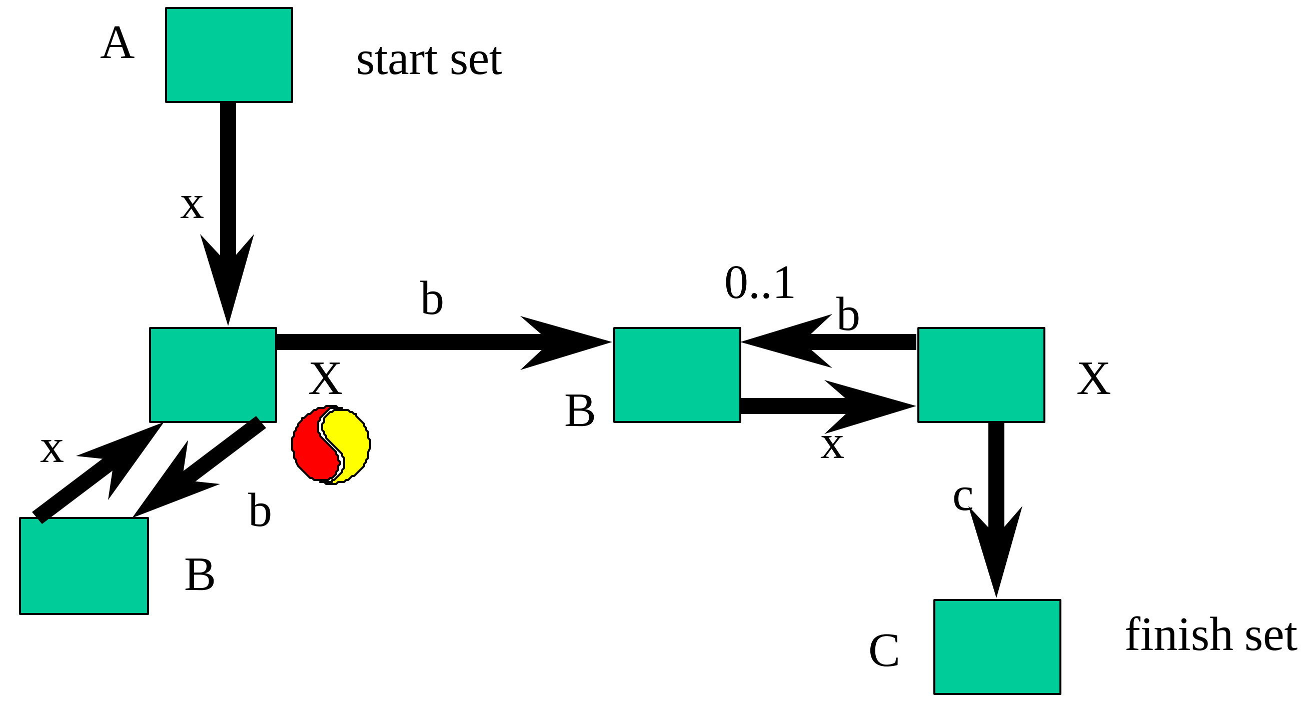
```
strategy:
{A -> B
 B -> C}
```

Object graph



After going back to x1:X

Traversal graph

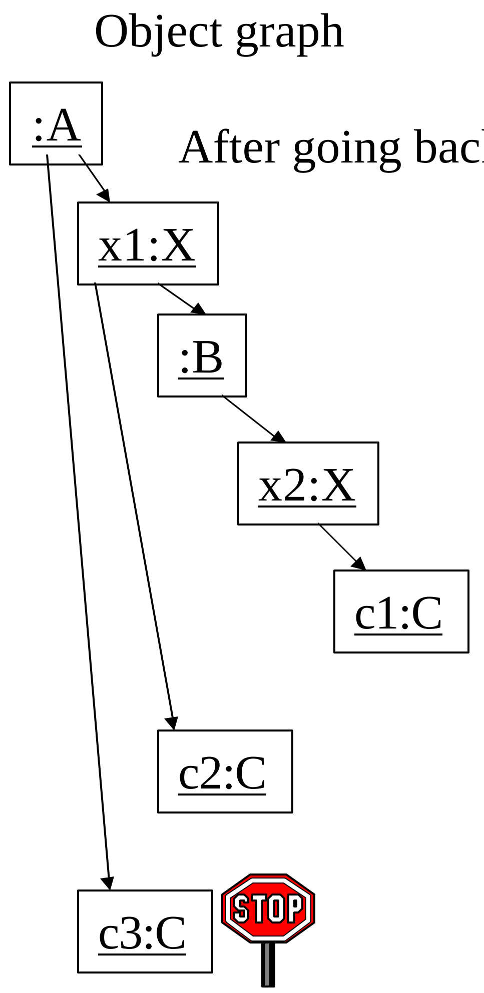


Used for token set and currently active object

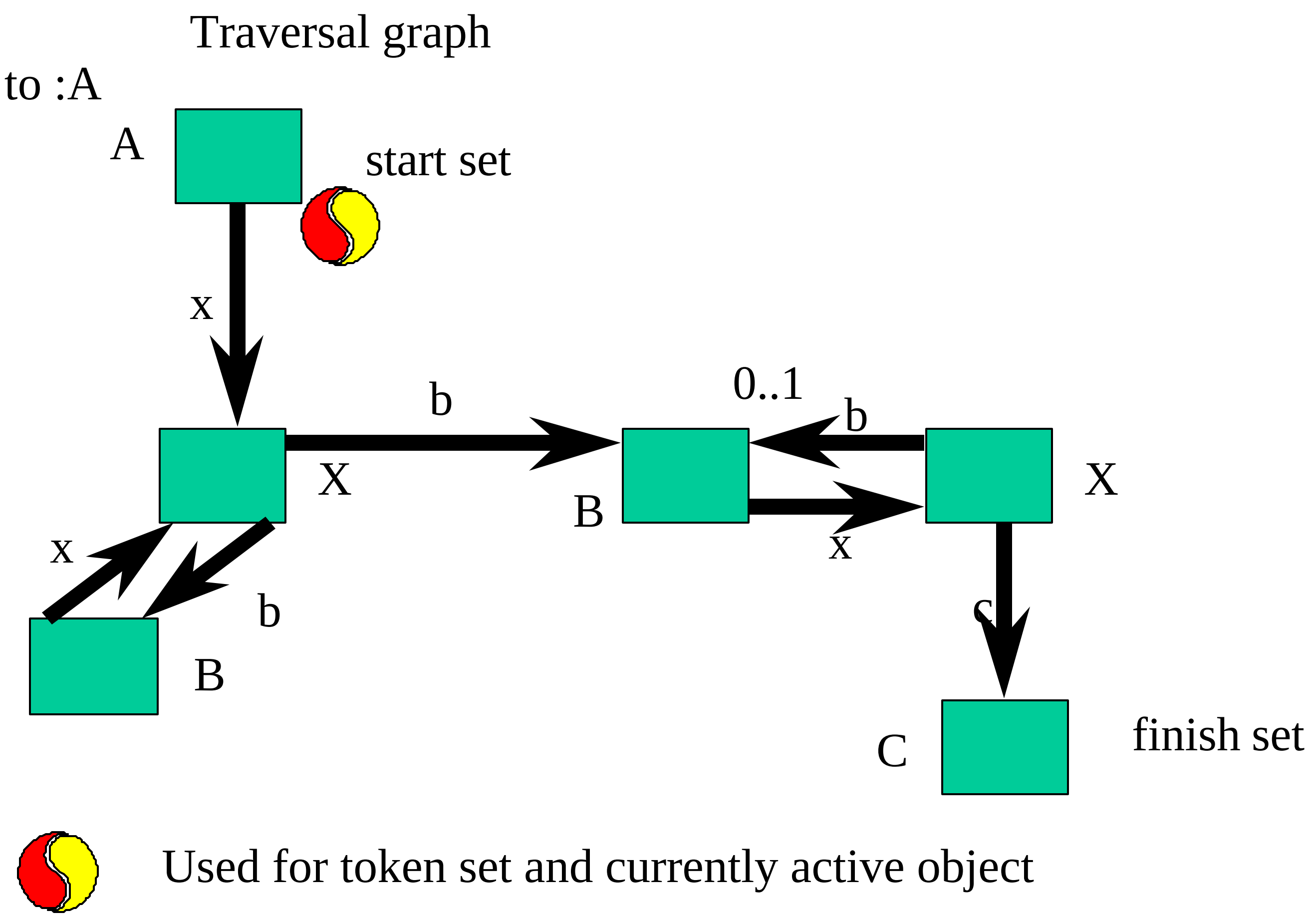
Active strategy graph edges: none `for :A`

Short-cut

```
strategy:
{A -> B
 B -> C}
```



After going back to :A



Used for token set and currently active object

SEdges

```
SEdges = Vector(SEdge).
SEdge =
  [<sourceName> String]
  [<targetName> String].
```

Class Dictionary

```
Person = Brothers Sisters
  Status.
Status : Single | Married.
Single = .
Married = <marriedTo> Person.
Brothers ~ {Person}.
Sisters ~ {Person}.
```

Strategy

```
from Person bypassing Person
  through Married
    bypassing Person
  through Person
    bypassing Person
  through {Brothers, Sisters}
    bypassing Person
to Person
```

Strategy and Traversal States

```
{ Person -> Married bypassing Person
  Married -> Person bypassing Person
  Person -> Brothers bypassing Person
  Person -> Sisters bypassing Person
  Brothers -> Person bypassing Person
  Sisters -> Person bypassing Person }
void before (Person h){
  if (previousSEdges(“”)) print(h.getName()); else
  if (previousSEdges(“{Married->Person}”))
    print(“spouse”+h.getName()); else
  if (previousSEdges(“{Brothers->Person}”))
    print(“brother-in-law”+h.getName());else
  if (previousSEdges(“{Sisters->Person}”))
    print(“sister-in-law”+h.getName());
}
```

Strategy and Traversal States

```
{ Person -> Married bypassing Person
  Married -> Person bypassing Person
  Person -> Brothers bypassing Person
  Person -> Sisters bypassing Person
  Brothers -> Person bypassing Person
  Sisters -> Person bypassing Person }
void before (Person h){
  if (nextSEdges(“Person->Married”)) print(h.getName()); else
  if (nextSEdges(“{Person->{Brothers,Sisters}}”))
    print(“spouse”+h.getName()); else
  if (previousSEdges(“{Brothers->Person}”))
    print(“brother-in-law”+h.getName());else
  if (previousSEdges(“{Sisters->Person}”))
    print(“sister-in-law”+h.getName());
}
```

Programming with Strategies

- At a given point during the traversal a set of strategy graph edges is active. The set of active strategy graph edges changes as we move through the object graph. When we are at a node in the object graph, we can talk about the next and previous set of strategy graph edges.
  - boolean nextSEdges(Strategy s);
  - boolean nextSEdges(String str);
  - boolean previousSEdges(Strategy s);
  - boolean previousSEdges(String str);
- The meaning is: is the set of strategy graph edges contained in the strategy?
- nextSEdges(“{Person->{Brothers,Sisters}}”)
- previousSEdges(“{Brothers->Person}”)

Programming with Strategies

- When we are at an edge in the object graph, we can talk about the current set of strategy graph edges.
  - boolean SEdges(Strategy s);
  - boolean SEdges(String str);
  - boolean SEdges(TraversalGraph tg);
- The meaning is: is the set of strategy graph edges contained in the strategy?

Visitor Methods for Strategy Edges

- void sbefore(Source s, Target t); // strategy
  - is executed before the traversal corresponding to `{s->t}` is started. More precisely, if `{s->t}` is currently not in the set of active strategy edges, but it will be after the next traversal step (going through an edge in the object graph), the code of the sbefore method will be executed before the next traversal step.
- void safter(Source s, Target t); // strategy
  - is executed after the traversal corresponding to `{s->t}` is finished. More precisely, if `{s->t}` is currently in the set of active strategy edges, but it will not be after the next traversal step (going through an edge in the object graph), the code of the safter method will be executed before the next traversal step.

Be more general

- Allow code attachment not only to strategy edges but to strategies.

Visitor Methods for Strategies

- void sbefore(Strategy s); // strategy
  - is executed before the traversal corresponding to s is started. More precisely, if the current object graph node is currently not in the scope of s, but it will be after the next traversal step (going through an edge in the object graph), the code of the sbefore method will be executed before the next traversal step.
- void safter(Strategy s); // strategy
  - is executed after the traversal corresponding to s is finished. More precisely, if s the current object graph node is currently in the scope of s, but it will not be after the next traversal step (going through an edge in the object graph), the code of the safter method will be executed before the next traversal step.

Follow DJ rule

- Wherever a strategy is used, a string may be used.
- void sbefore(String str); // strategy
- void safter(String str); // strategy

Programming with strategies

check whether currently in scope of subtraversal

// may be used in before, cbefore, rbefore, sbefore

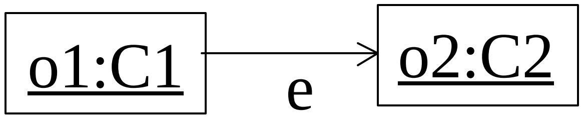
- // sg a substrategy of current strategy

```
if (SEdges(sg)) {
    // currently in traversal determined by strategy sg
```

- // tg a subgraph of current traversal graph

```
if (SEdges(tg)) {
    // currently in traversal determined by tg
```

Traversing an edge: From C1 to C2



declared type of o2 is C3=>C2

go down e iff C1 <=.C C3 => C2

when we go down e, we execute the before method of all cbefore methods that match the edge e in the order the cbefore methods are listed in the visitor.

Similarity

- When we traverse an edge in the object graph, several visitor methods (for edges) will be executed.
  - the same visitor may have several cbefore methods whose expansion includes the same object graph edge.
- When we traverse a node in the object graph, several visitor methods (for nodes) may be executed.
  - when an object graph node method is executed we also execute the methods of the super classes.

Executing cbefore_v(F f, W w)?

- Assuming that only one edge matches
- Object graph edge labeled v

```
o1:SubClass(F) -> o2:SubClass(W)
```

- Is F a superclass of class(o1)?
- Has F a construction edge labeled w to W?
- Execute method.

Finding the edge methods

- Object graph edge labeled v

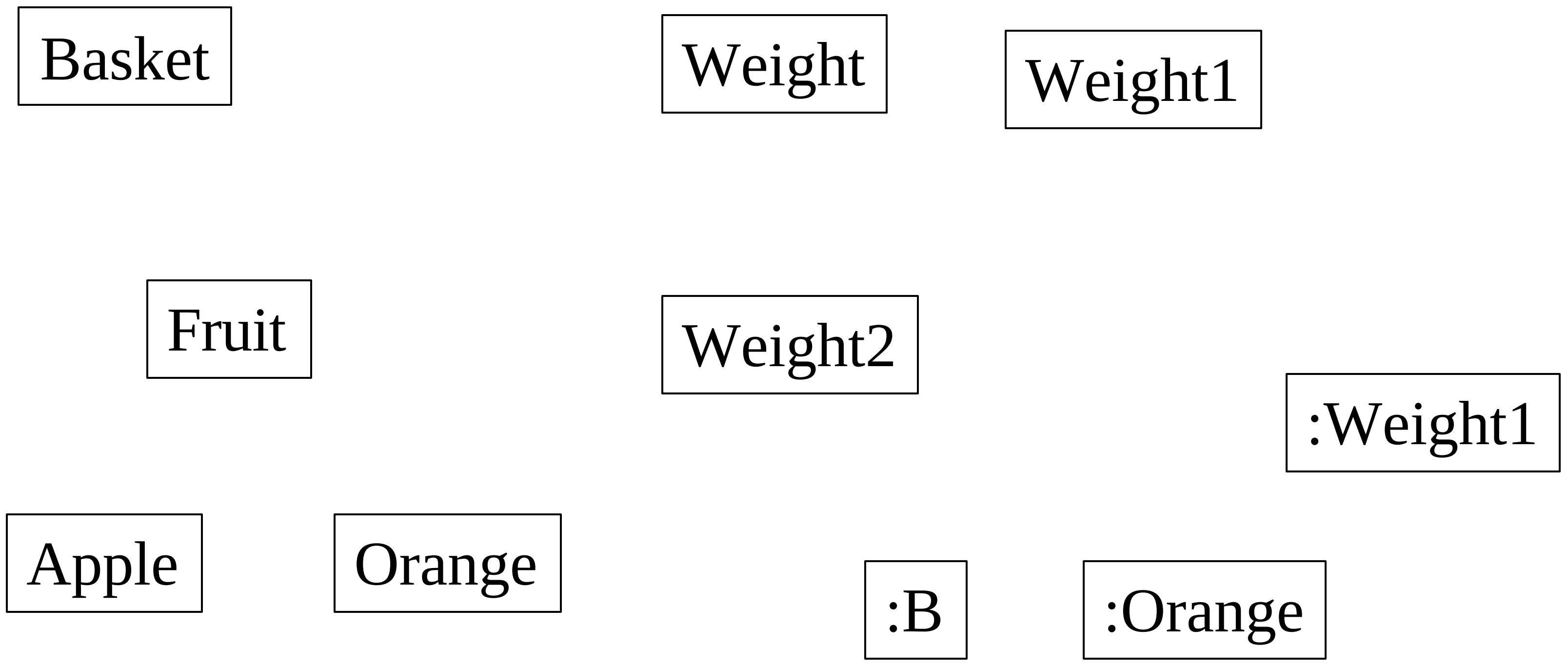
  `o1:class(o1) -> o2:class(o2)`
- Is there any edge visitor cbefore_v for a superclass of class(o1)?
- Execute method.
- Use a hash table.

before_v(F f, W w)?

- The second argument (W w) is uniquely determined by the class graph and is only given so the programmer can use the interface of the target node.
- Semantic check done by DJ: Is W correct? Executed each time the before method is called?

Difference to DemeterJ for edge methods

- DJ only allows edge methods on construction edges and repetition edges while DemeterJ also allows edge methods on inheritance edges.
- Motivation: In DJ, we want the behavior of a program to be invariant under flattening of the class graph.


Basket
Weight
Weight1
Fruit
Weight2
:Weight1
Apple
Orange
:B
:Orange