Modelling Workflow Systems with Active Folders:

A Decentralized and Declarative approach to Collaborative Systems with an emphasis on the Artifacts and the Stakeholders

Eric Badouel

INRIA - Rennes

June 2012

Plan

Document-centric Workflow Systems

Of documents and men

Business Process Management

- Workflows systems are traditionnally organized as a set of coordinated activities between the various stakeholders of the system.

Business Process Management

- Workflows systems are traditionnally organized as a set of coordinated activities between the various stakeholders of the system.
- In this context the emphasis is on the flow of tasks, which is usually modeled using centralized and state-based formalisms like automata, or Petri nets.

Business Process Management

- Workflows systems are traditionnally organized as a set of coordinated activities between the various stakeholders of the system.
- In this context the emphasis is on the flow of tasks, which is usually modeled using centralized and state-based formalisms like automata, or Petri nets.
- Data that are exchanged during the processing of a task play a secondary role when they are not simply ignored.

Business Process Management

- Workflows systems are traditionnally organized as a set of coordinated activities between the various stakeholders of the system.
- In this context the emphasis is on the flow of tasks, which is usually modeled using centralized and state-based formalisms like automata, or Petri nets.
- Data that are exchanged during the processing of a task play a secondary role when they are not simply ignored.
- Similarly, stakeholders are used as plain ressources of these systems like a machine or a robot in a production line.

Data-centric Workflows

- By contrast, the more recent model of data-centric workflow systems, put forward by IBM, puts stress on the exchanged documents, the so-called Business Artifacts.

Data-centric Workflows

- By contrast, the more recent model of data-centric workflow systems, put forward by IBM, puts stress on the exchanged documents, the so-called Business Artifacts.
- An artifact is a document that conveys all the informations related to a given task from its inception in the workflow till its completion.

Data-centric Workflows

- By contrast, the more recent model of data-centric workflow systems, put forward by IBM, puts stress on the exchanged documents, the so-called Business Artifacts.
- An artifact is a document that conveys all the informations related to a given task from its inception in the workflow till its completion.
- However this model is again state-based (the life cycle of an artifact is given by an automaton) and centralized, and stakeholders are still not first class citizens.

Active Folders

- A decentralized and purely declarative approach to

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

Our proposition

- A modular language-based approach for the design of the end-user workspace.

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

Our proposition

- A modular language-based approach for the design of the end-user workspace.
- This workspace is modeled by a so-called Active Folder which combines structure, data, and computations.

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

Our proposition

- A modular language-based approach for the design of the end-user workspace.
- This workspace is modeled by a so-called Active Folder which combines structure, data, and computations.
- The latter are given by semantic rules used to derive data values from contextual information in the structure –as in a spreadsheet system– or to restrict user interactions to guide him in the completion of some required data fields.

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

Our proposition

- A modular language-based approach for the design of the end-user workspace.
- This workspace is modeled by a so-called Active Folder which combines structure, data, and computations.
- The latter are given by semantic rules used to derive data values from contextual information in the structure –as in a spreadsheet system– or to restrict user interactions to guide him in the completion of some required data fields.
- This model use a variant of attribute grammars

Active Folders

- A decentralized and purely declarative approach to
- collaborative systems which are at the same time user-centric and data sensitive.

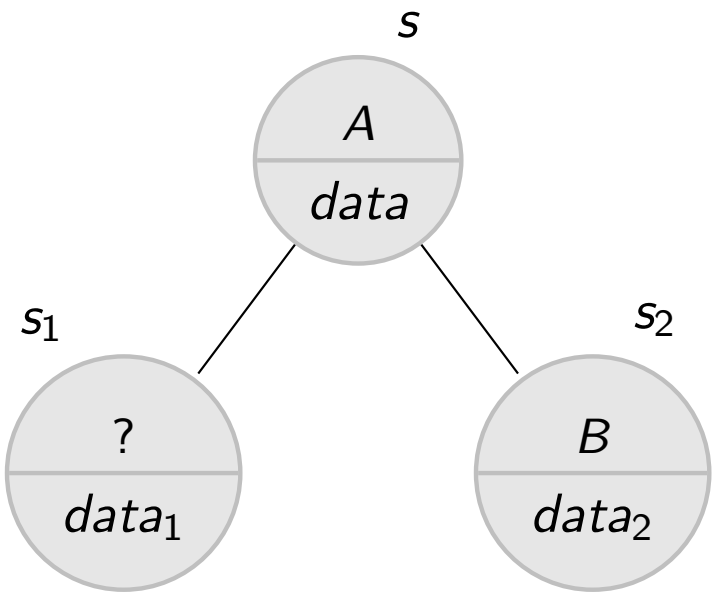
Our proposition

- A modular language-based approach for the design of the end-user workspace.
- This workspace is modeled by a so-called Active Folder which combines structure, data, and computations.
- The latter are given by semantic rules used to derive data values from contextual information in the structure –as in a spreadsheet system– or to restrict user interactions to guide him in the completion of some required data fields.
- This model use a variant of attribute grammars
- The actions in the system are then modelled as the productions of the grammar (used for refining a budding node of an artifact).

The Model of Active Folders

A Case Study: The Editorial Process of an Electronic Journal

Artefact



Grammar

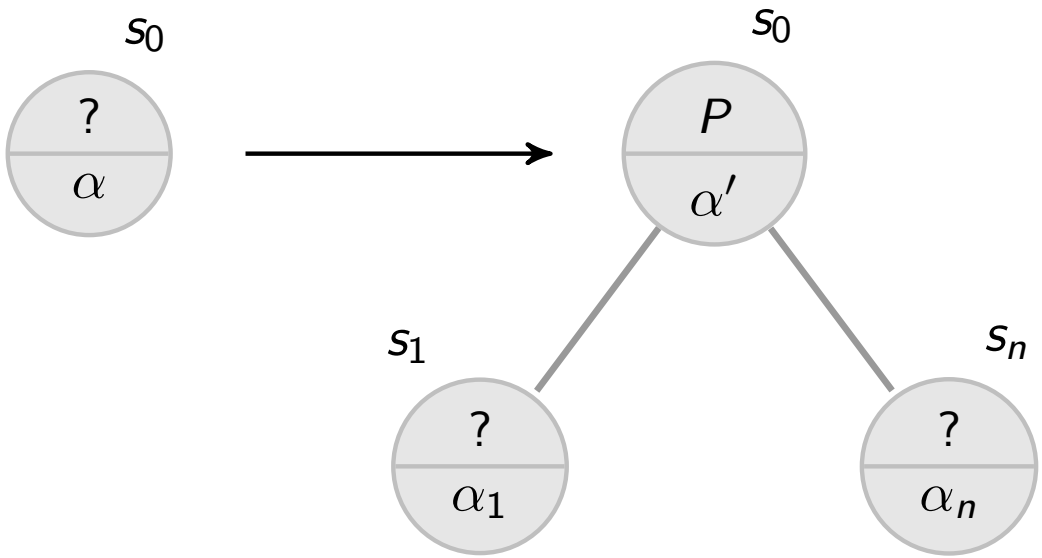
$$A : s \longrightarrow s_1 \; s_2$$
$$B : s_2 \longrightarrow$$

local information: $\tau ::= (\tau_1, \ldots, \tau_n) \mid \tau^* \mid \mathbf{b}$

```
article =  {date :: Date;
            title :: String;
            correspondingauthor :: author;
            coauthors :: author*;
            abstract :: String;
            text :: PDF}
```

Definition of a node: Production

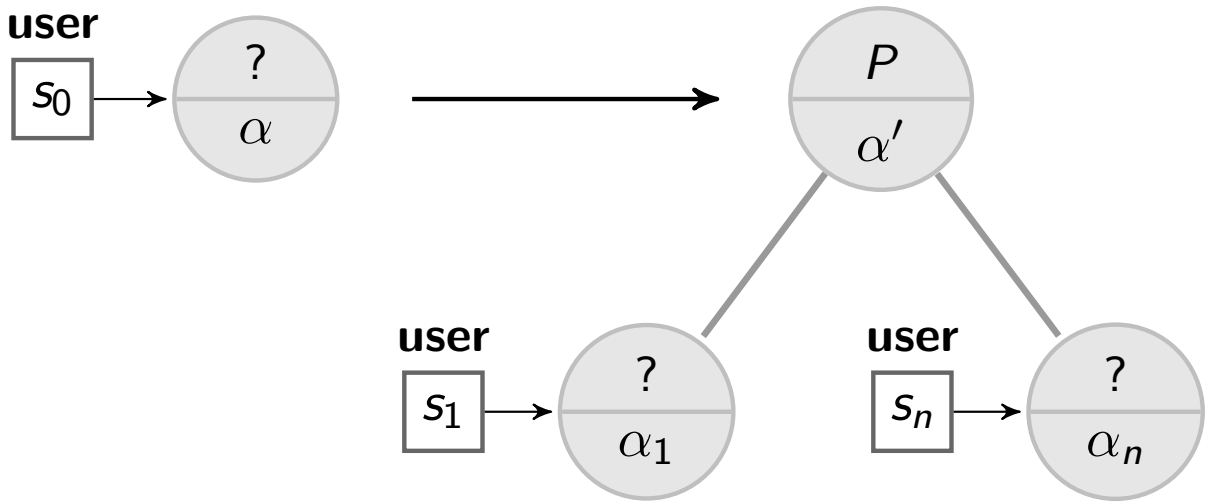
$P : \langle x_0 :: s_0 \rangle \longrightarrow \langle x_1 :: s_1 \rangle \cdots \langle x_n :: s_n \rangle$



Local information is

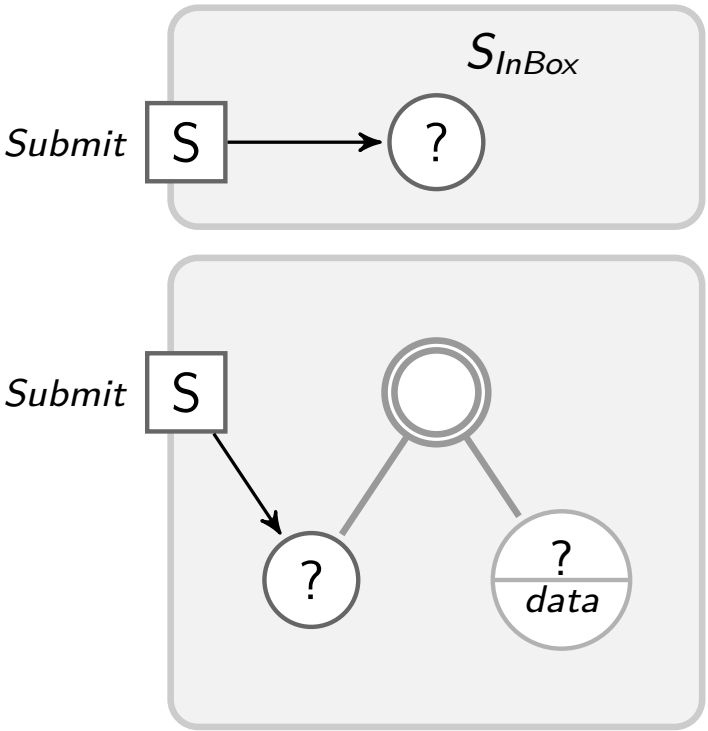
- initialized at the creation of the node
- updated when it is defined by a production.

Active Documents: Attaching programs to open nodes



- A generator serves only once (to define the node it is attached to) but new generators must be attached to the newly created (open) nodes.
- A generator is a reactive program triggered either
  - by user interactions
  - by the arrival of a value in an input port (public channel)
  - by reading a value of an input node (that should be linked to it beforehand).

Submitting a paper



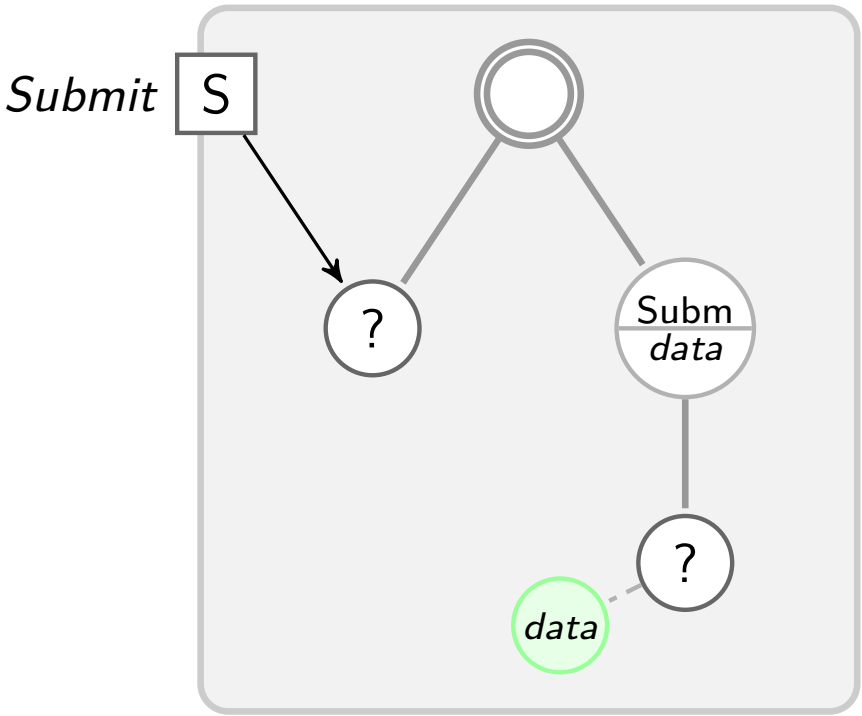
$$S(port, val) : \langle x :: S_{InBox} \rangle \rightarrow \langle x' :: S_{InBox} \rangle \ \langle y :: elet \rangle$$

where $x' = S(port)$

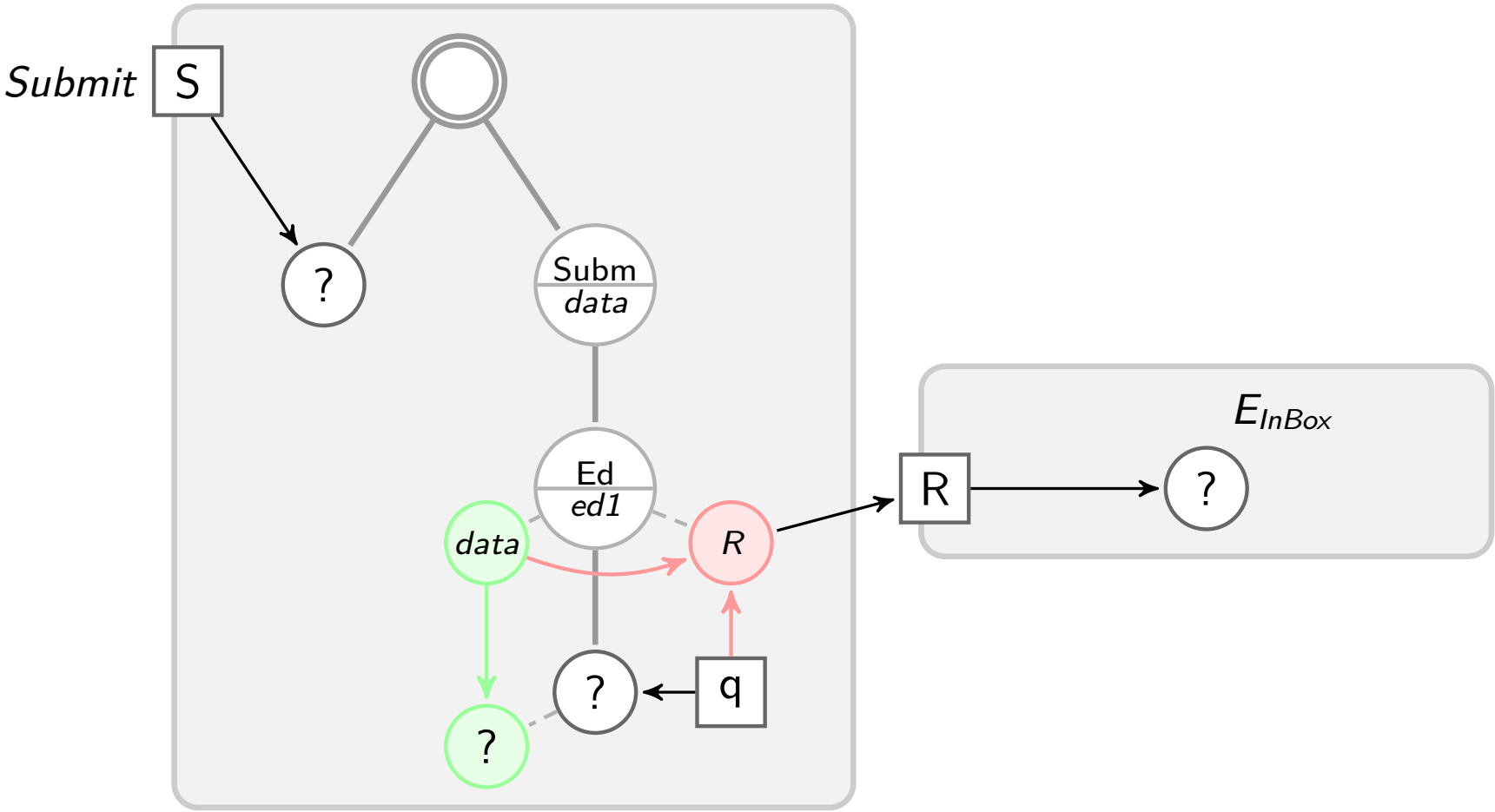
$y = \mathbf{user}_s$

$y.local = val$

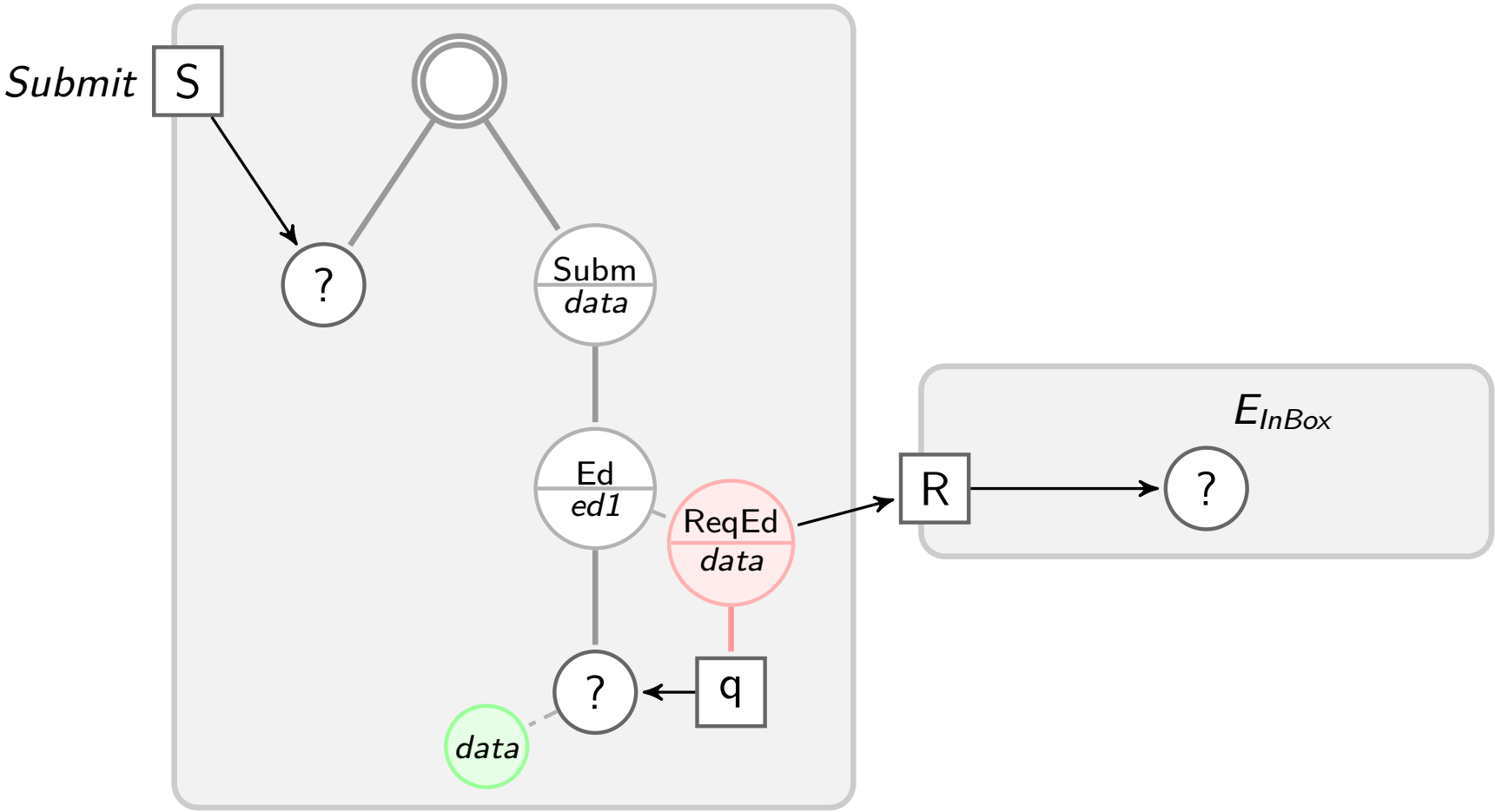
Submission validated by the Editor in Chief



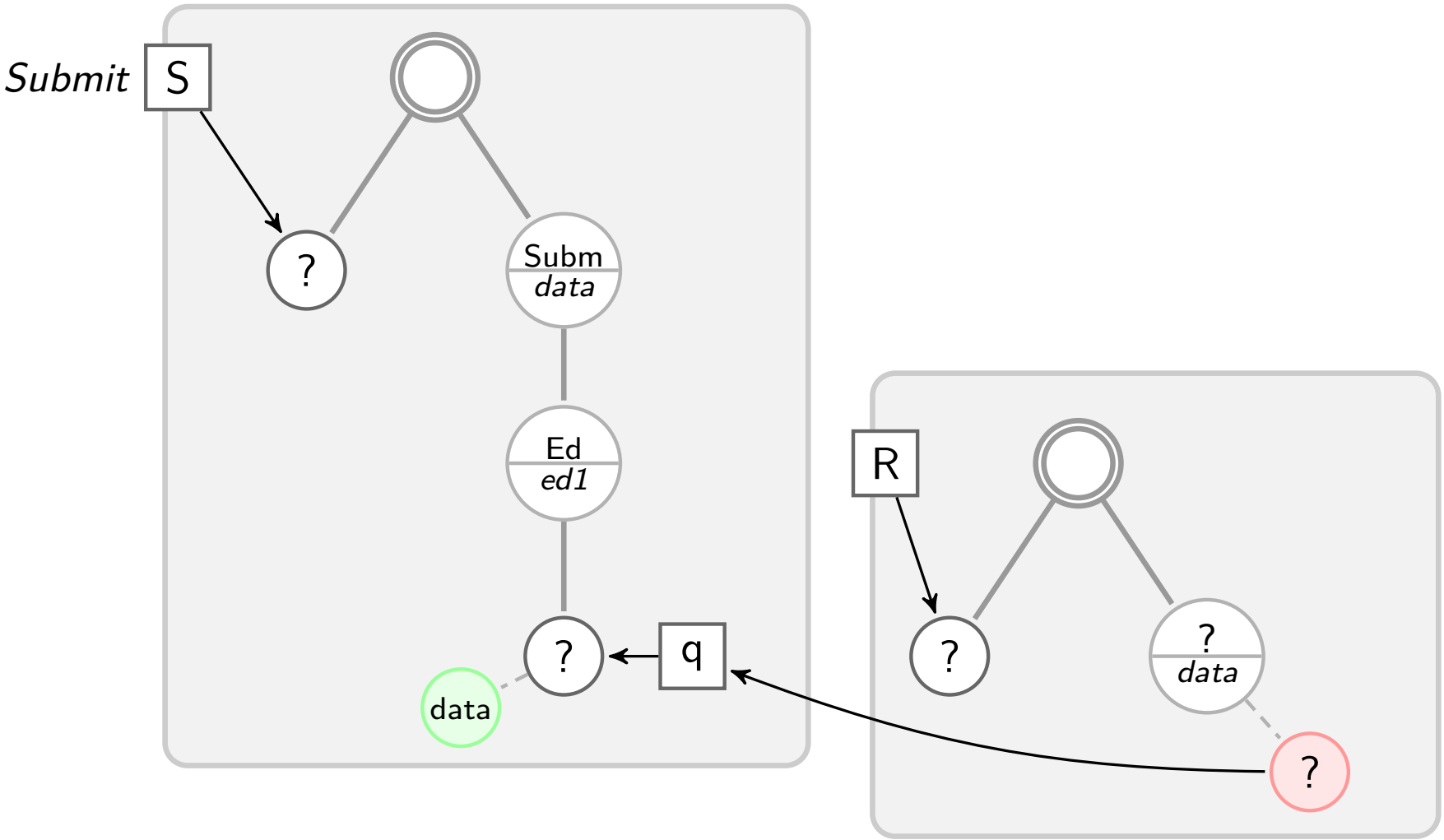
Sollicitation of an Associate Editor



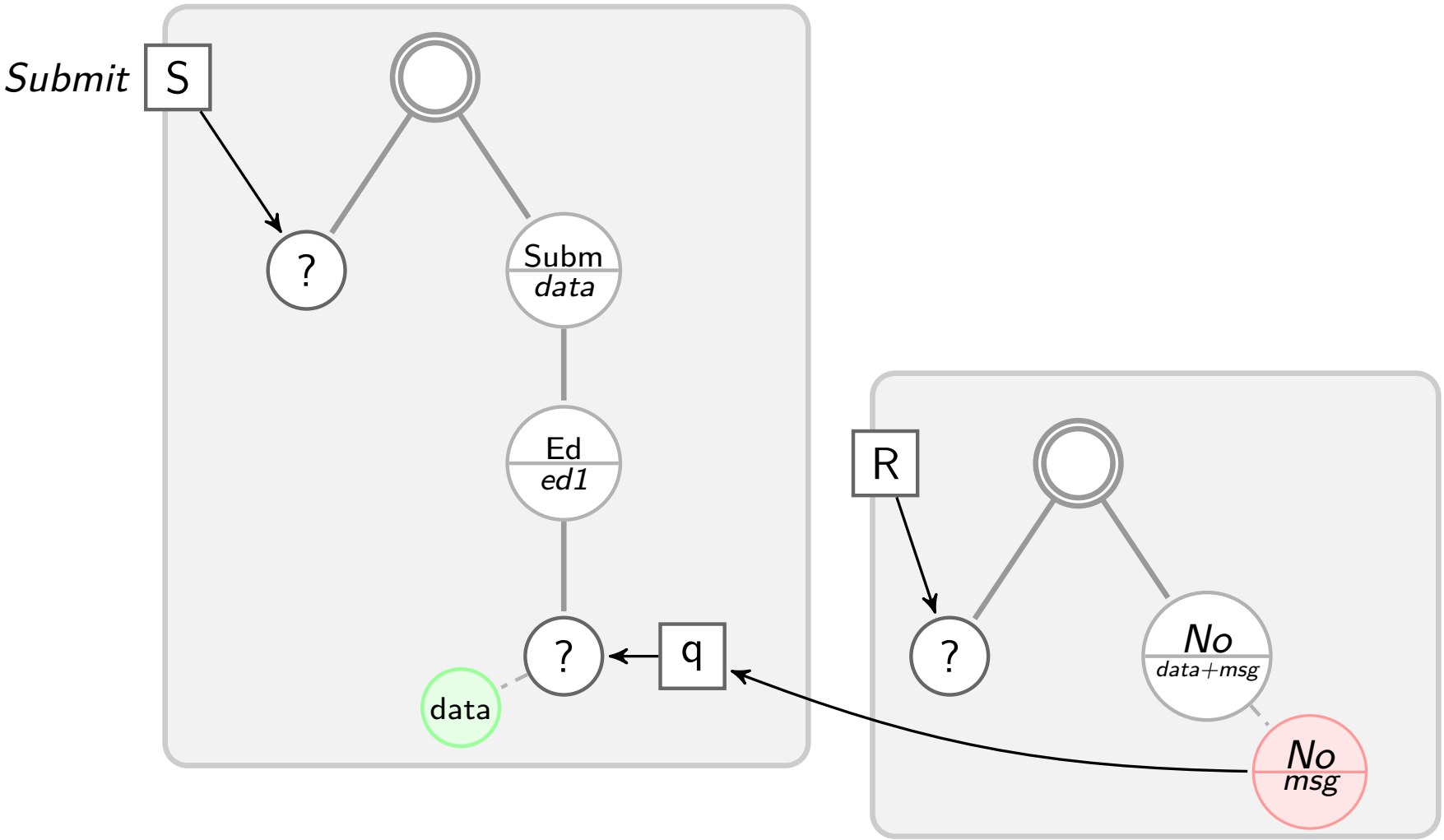
Sollicitation of an Associate Editor (2)



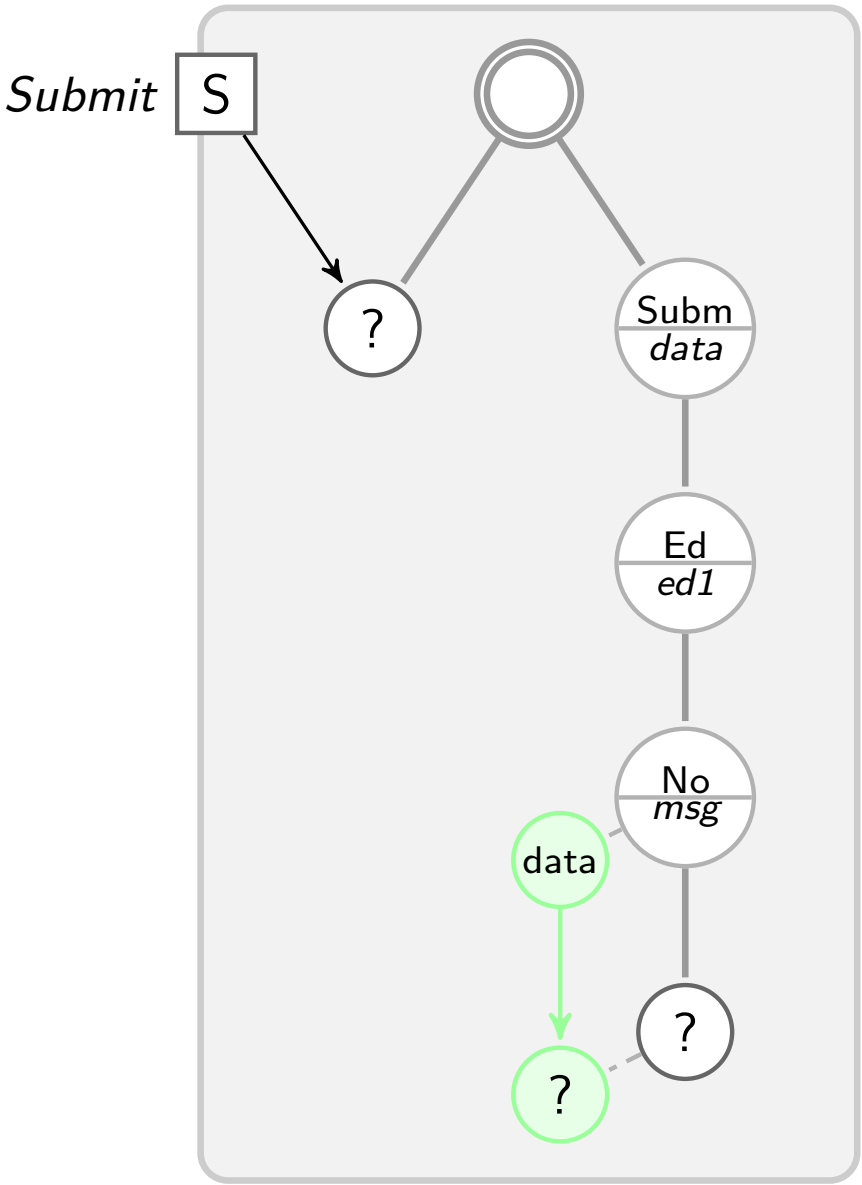
Sollicitation of an Associate Editor (3)

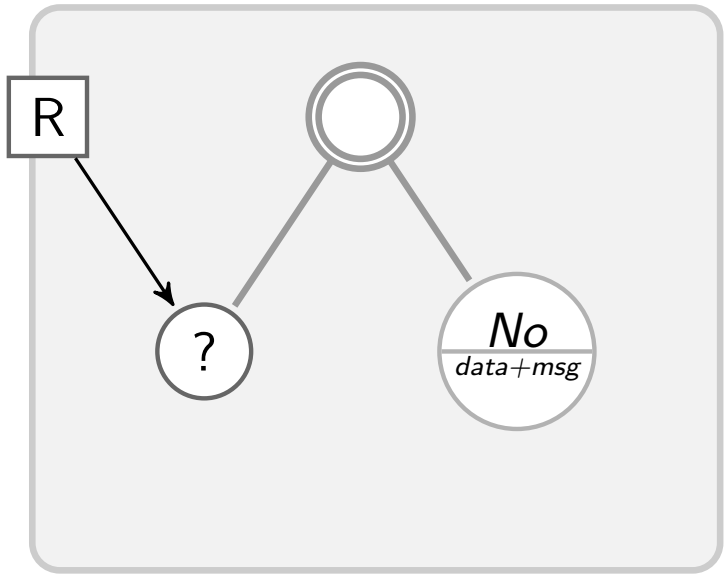


Associate Editor's refusal

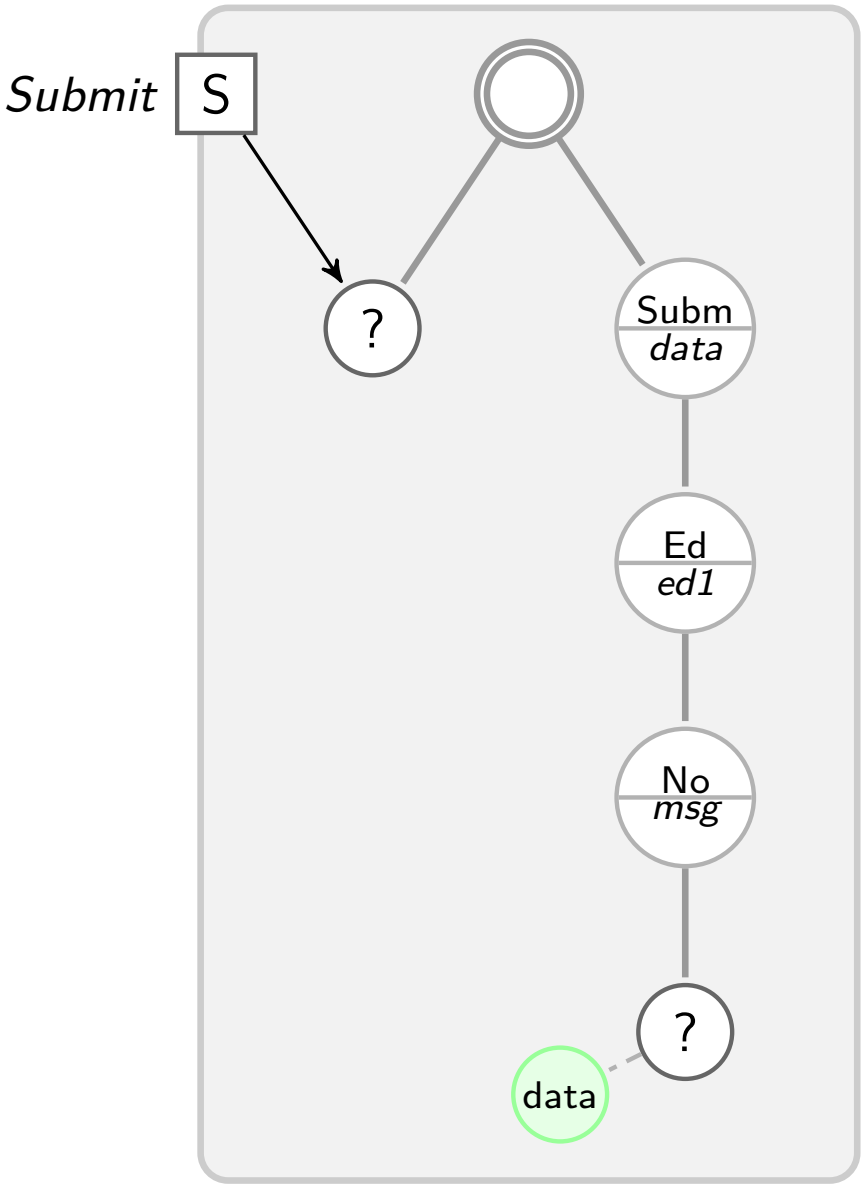


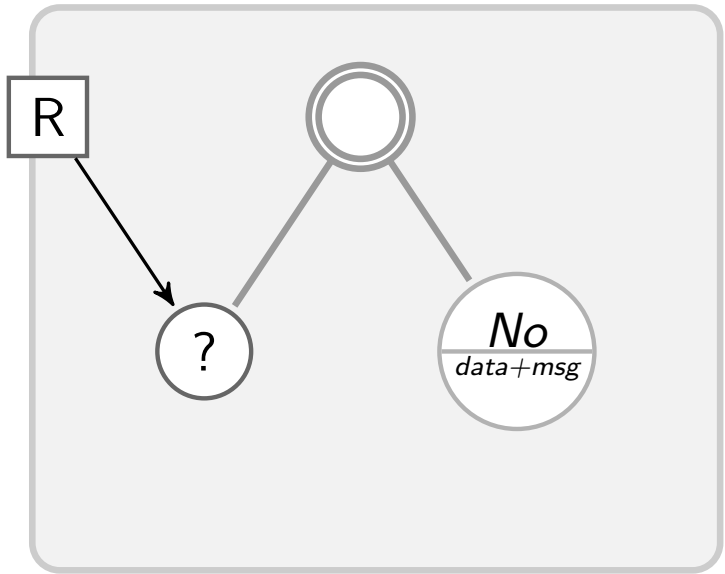
Associate Editor's refusal (2)



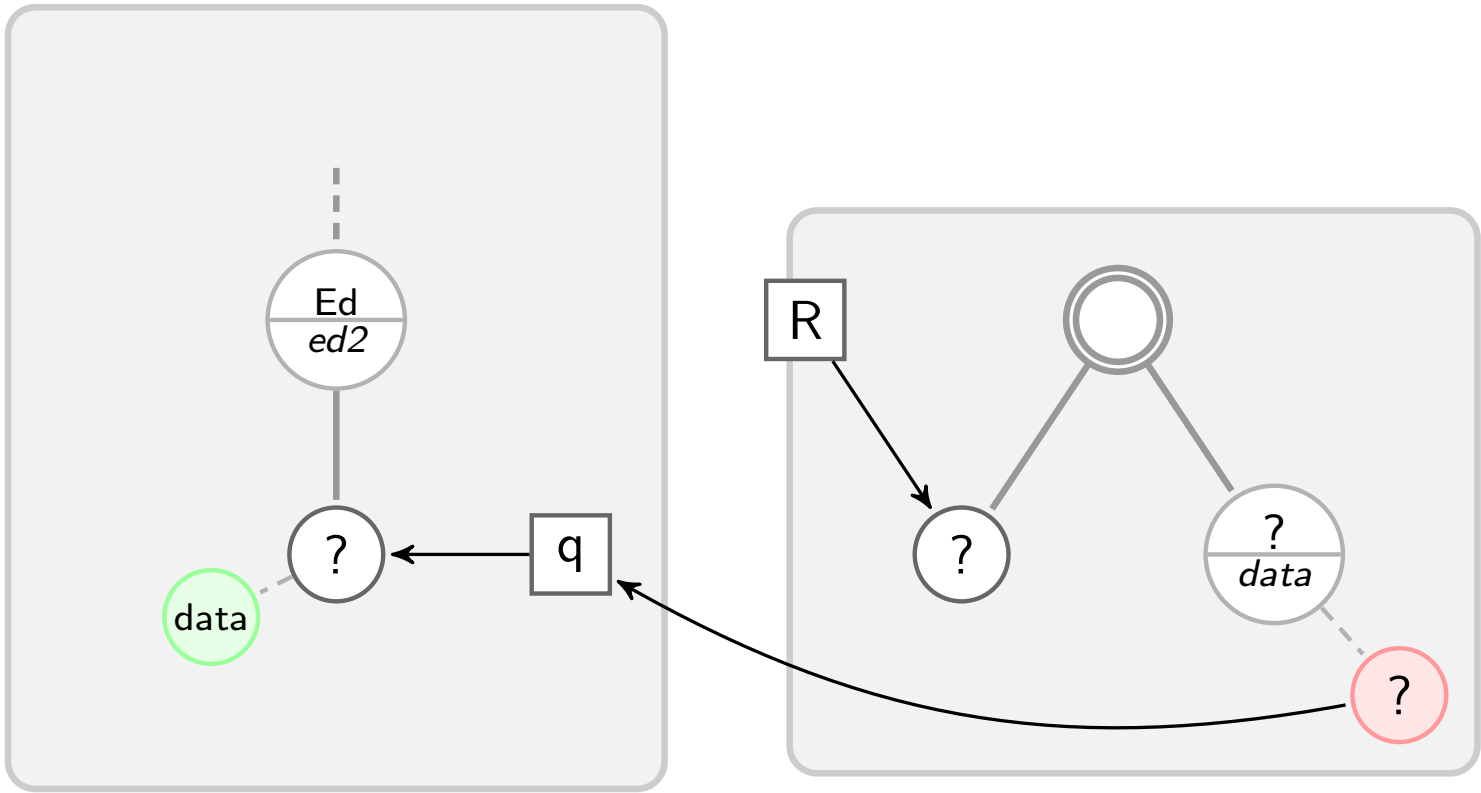


Ready to look for another Associate Editor

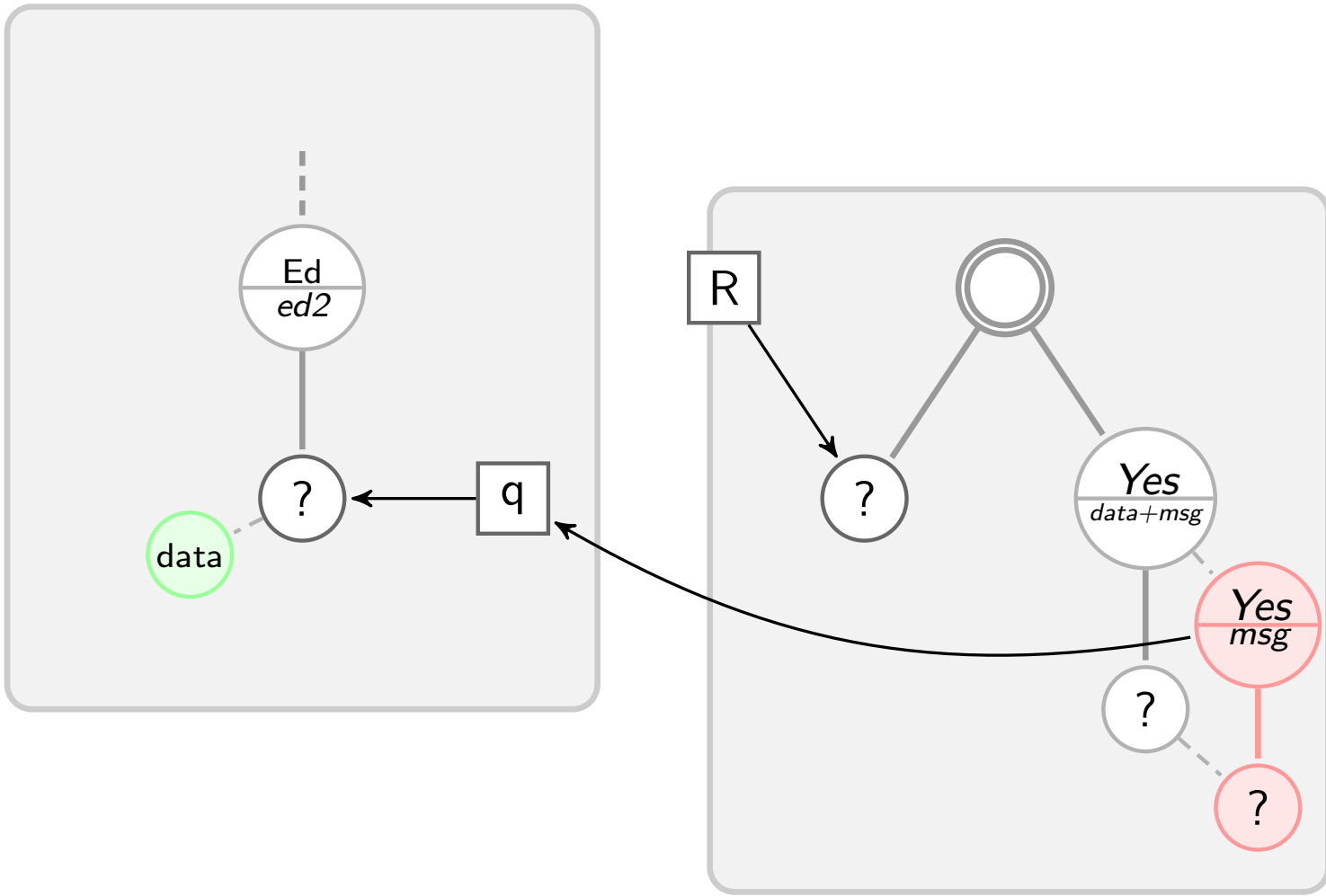




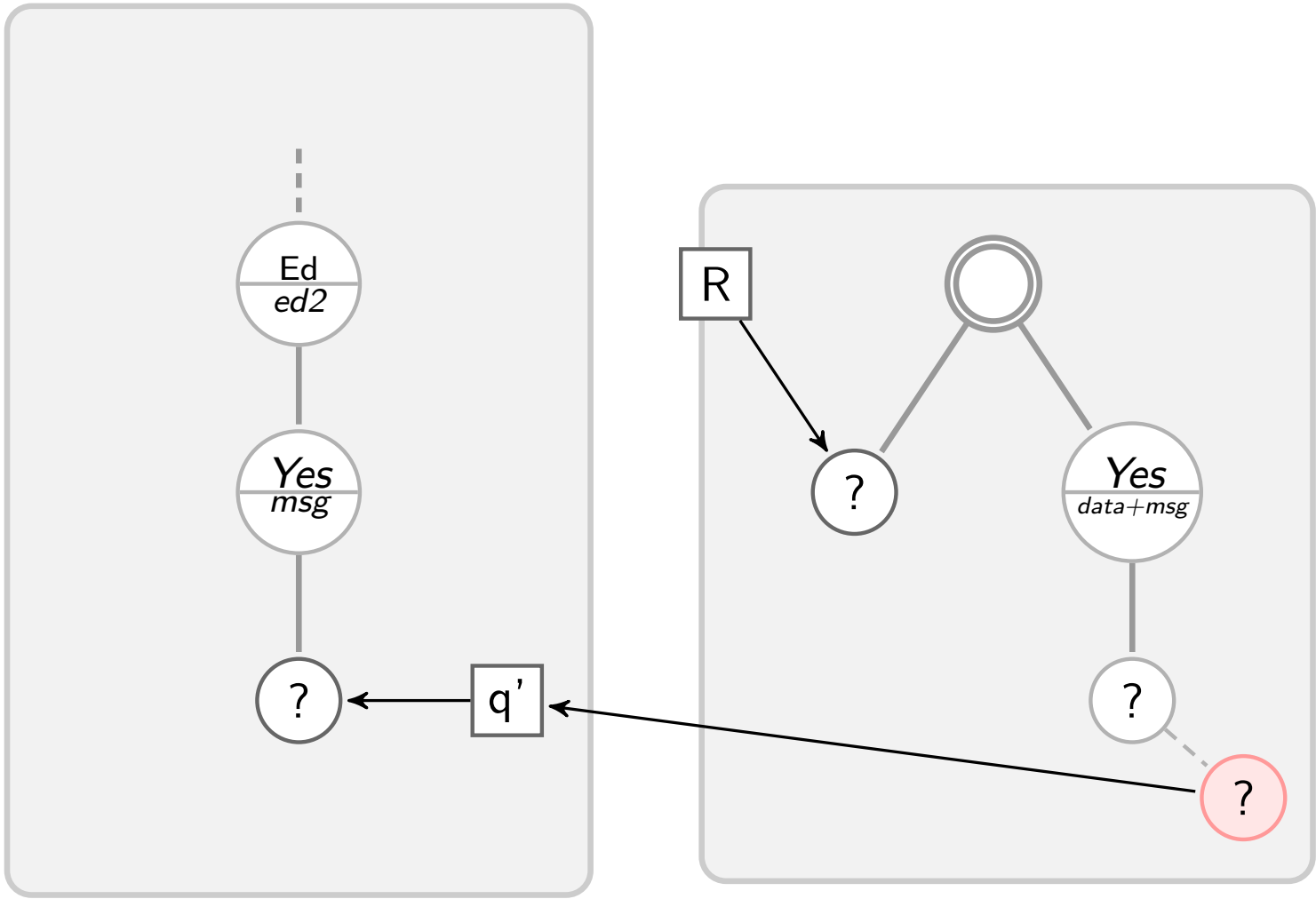
Editor 2 has been sollicited



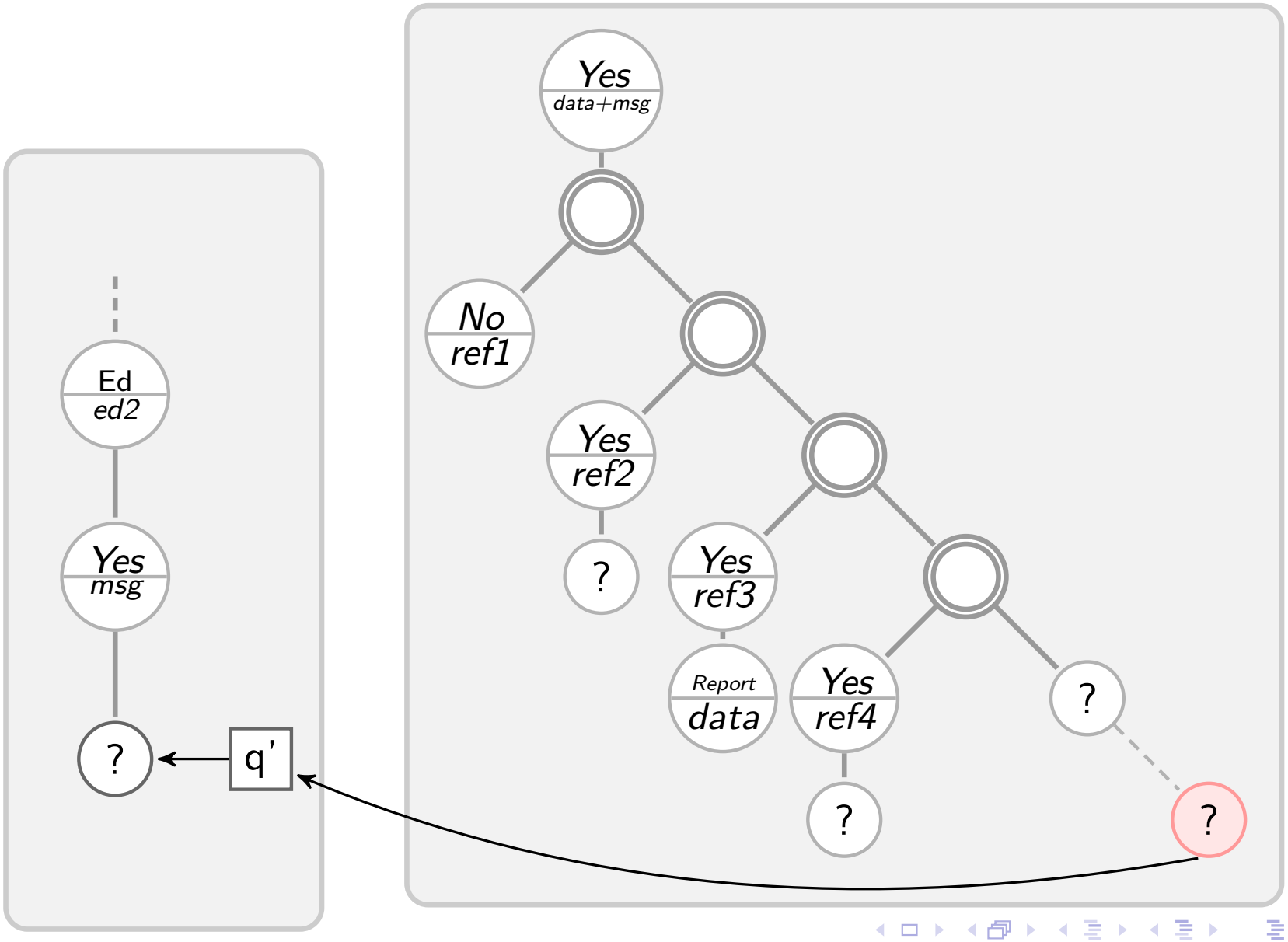
Editor 2 has accepted



Editor 2 has accepted (2)



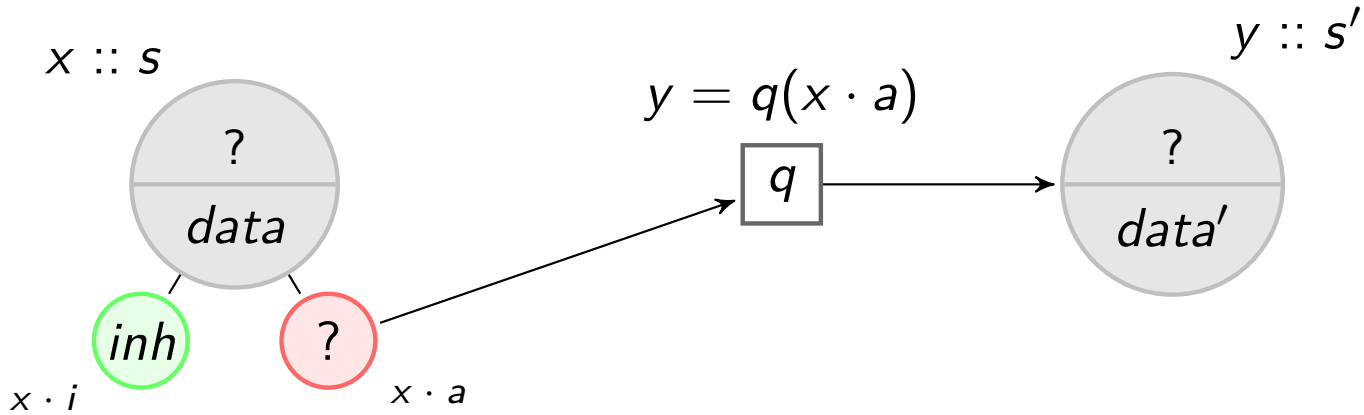
Some time later ...



Attributes

Attributes contain (structural) information attached to nodes which are used to compute some synthesized information to be send to an external node having subscribed for this information.

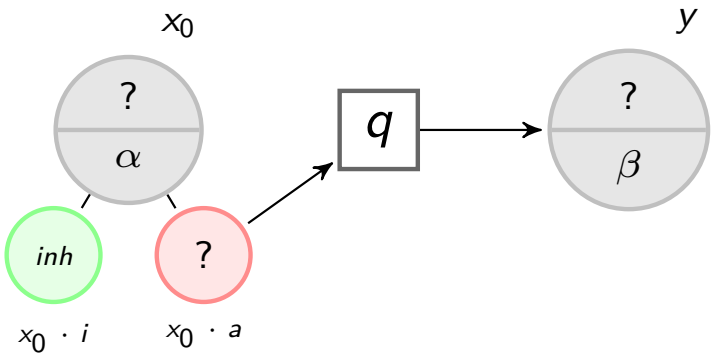
place y subscribes to a view q of the synthesized attribute $a \in Syn(s)$ of node $x :: s$



attributes vs local information

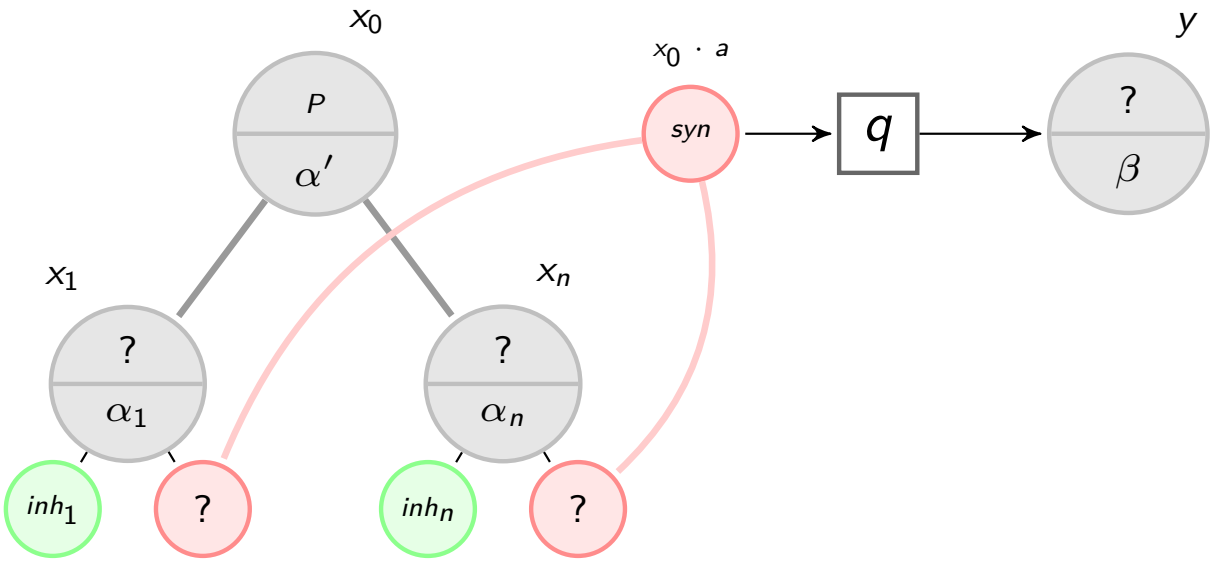
Attributes are produced and consummed (by semantics rules), they are attached to open nodes (buds) and therefore this auxiliary information (carried by attributes) no longer exists when the artefact is completed (no open nodes left).

Semantics rules

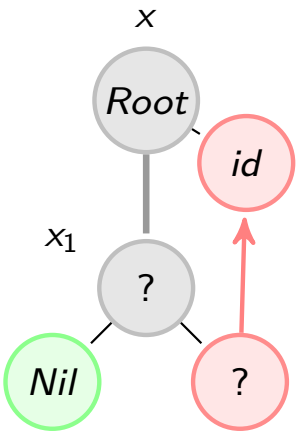


$$z = t(z_1, \cdots, z_m)$$

- t is a tree over an auxiliary alphabet of constructors
- copy rule when $t = z'$ is a variable
- used variables:
  $\{x_0 \cdot i \mid i \in Inh(s_0)\} \cup_{1 \le j \le n} \{x_j \cdot a \mid a \in Syn(s_j)\}$
- defined variables:
  $\{x_0 \cdot a \mid a \in Syn(s_0)\} \cup_{1 \le j \le n} \{x_j \cdot i \mid i \in Inh(s_j)\}$

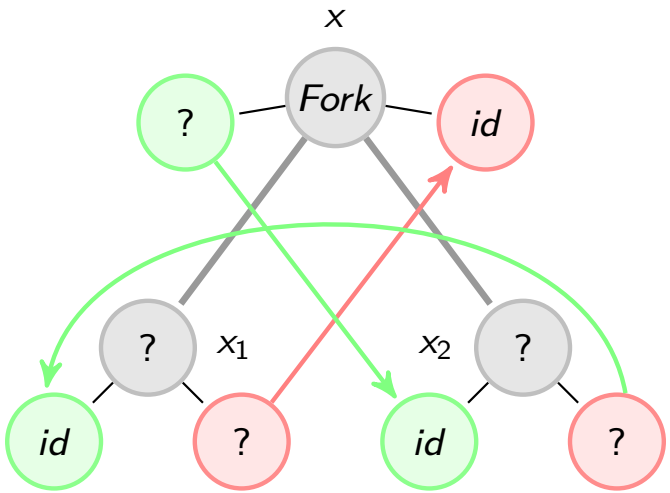


An Example: flattening a binary tree



$$Root : \langle x :: Root \rangle \rightarrow \langle x_1 :: Bin \rangle$$

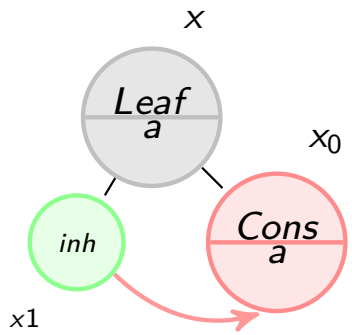
where $x \cdot s = x_1 \cdot s$
$x_1 \cdot h = Nil$



$$Fork : \langle x :: Bin \rangle \rightarrow \langle x_1 :: Bin \rangle \ \langle x_2 :: Bin \rangle$$

where $x \cdot s = x_1 \cdot s$
$x_1 \cdot h = x_2 \cdot s$
$x_2 \cdot h = x \cdot h$

An Example: flattening a binary tree (2)

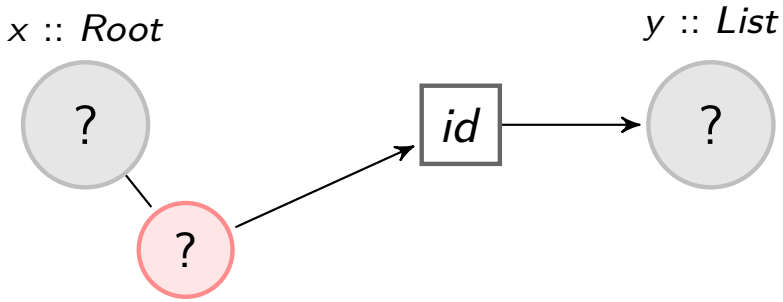


$$Leaf(a) : \langle x :: Bin \rangle \rightarrow$$

where $x.local = a$

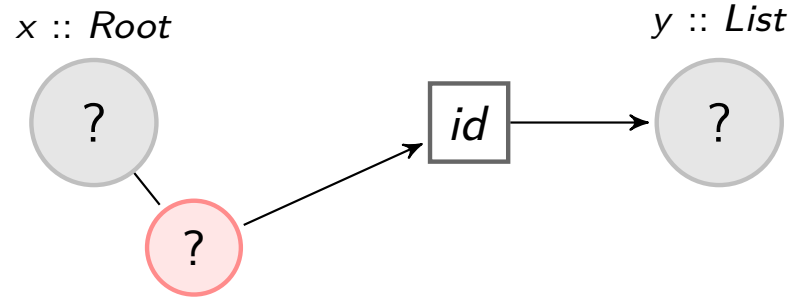
$x \cdot s = Cons_a(x \cdot h)$

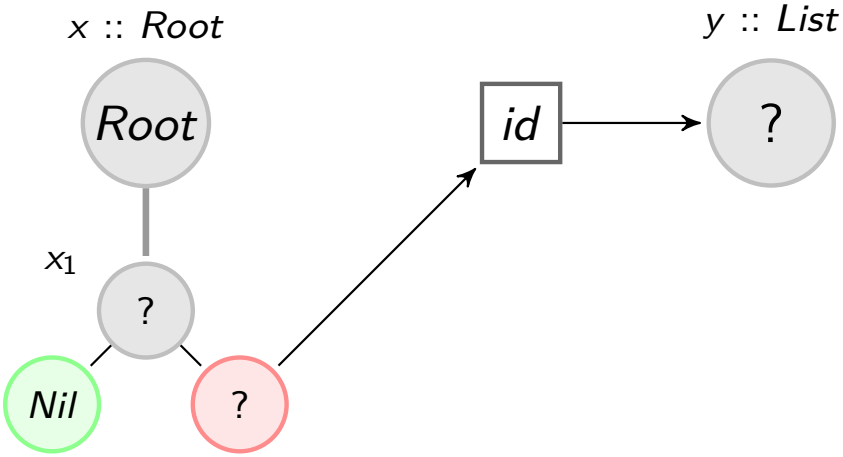
Initial configuration



$$y = x \cdot s$$

Applying the semantic rules





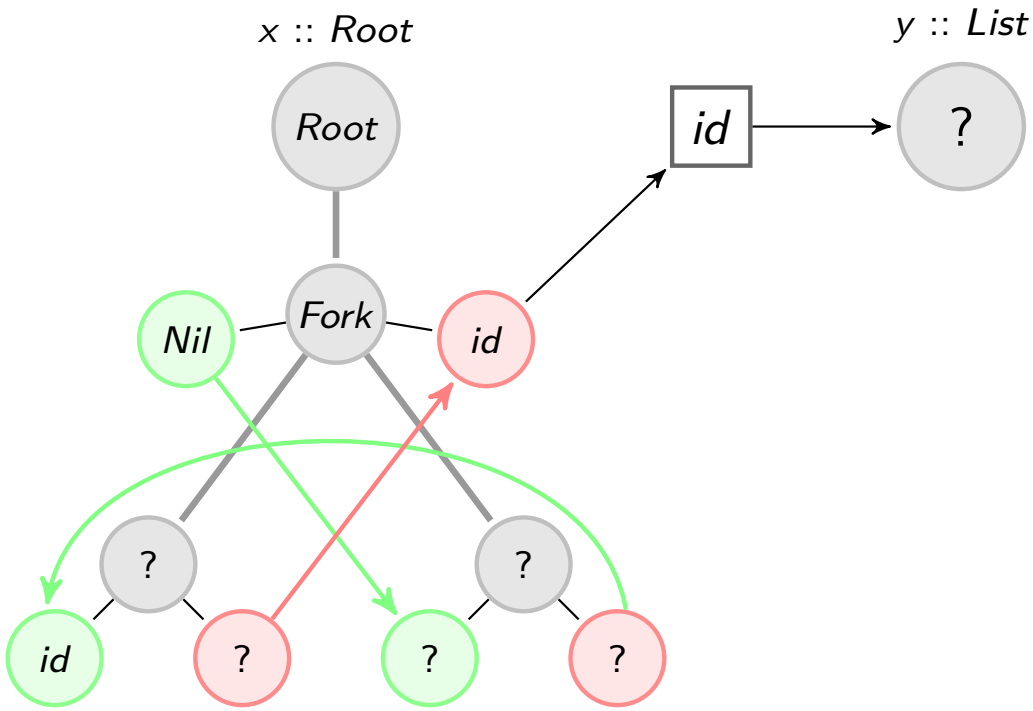
To apply production *Root* at node x

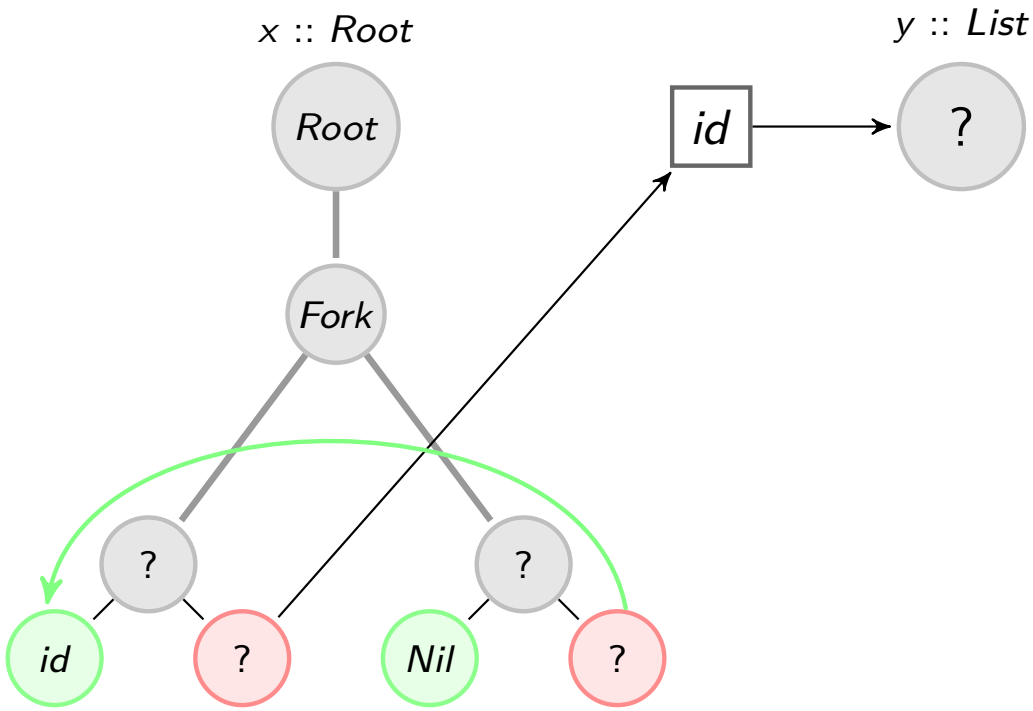
- Add to the current configuration $y = x \cdot s$
- the equations of the production where x is substitued to the left-hand side variable and the variables of the right and side (and their associated attributes) are fresh names

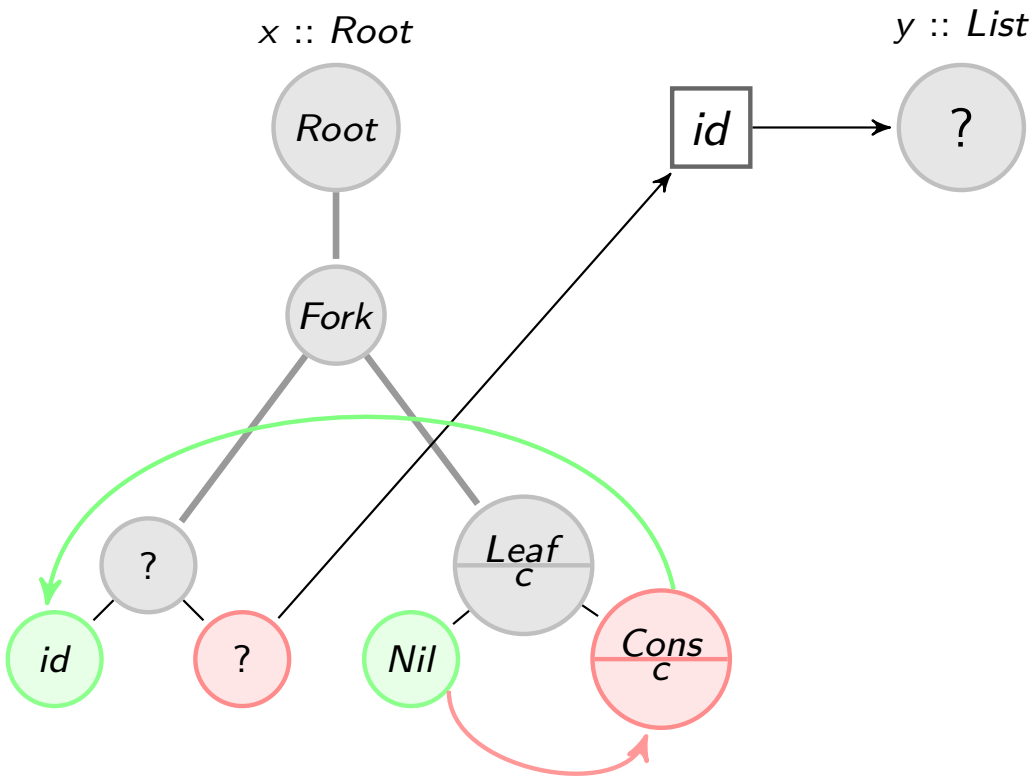
$$\begin{array}{lcl} x \cdot s & = & x_1.s \\ x_1.h & = & Nil \end{array}$$

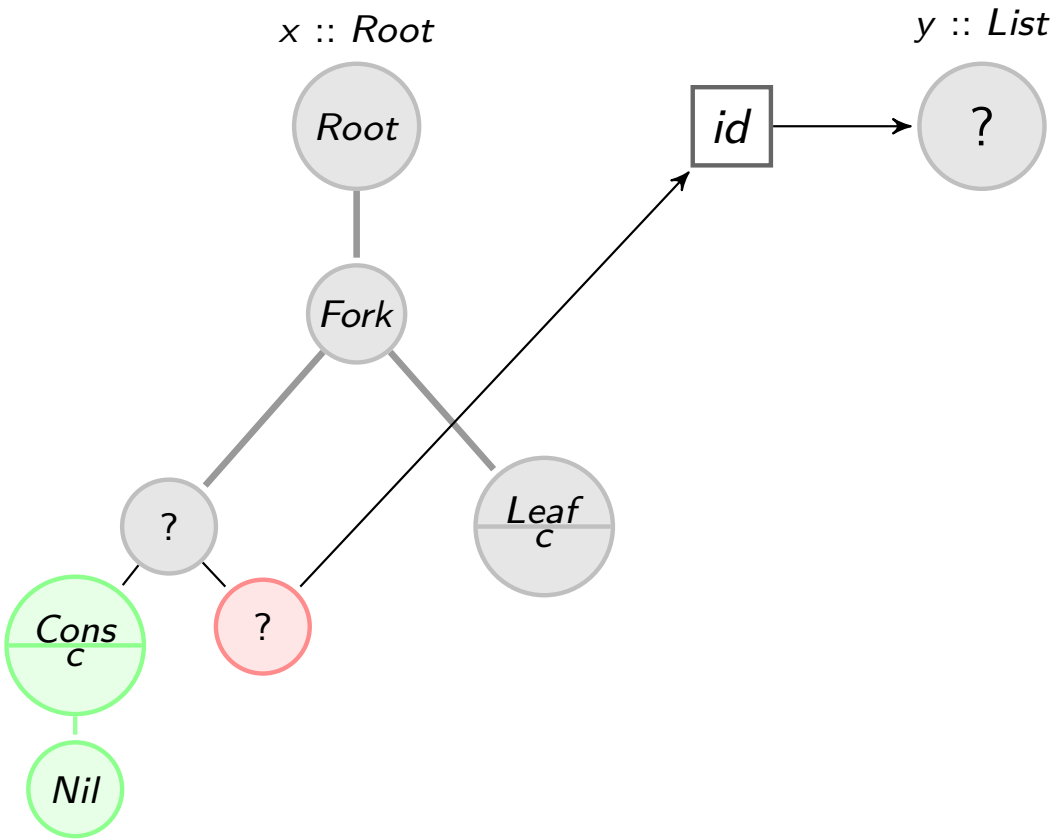
- The new configuration is obtained after
  - elimination of the copy rules: elimination of a variable z which is both used and defined after replacingeach of its used occurrences by its definition
  - elimination of the equations that define an inherited attribute of x or use a synthesized attribute of x that may remain after elmination of the copy rules.

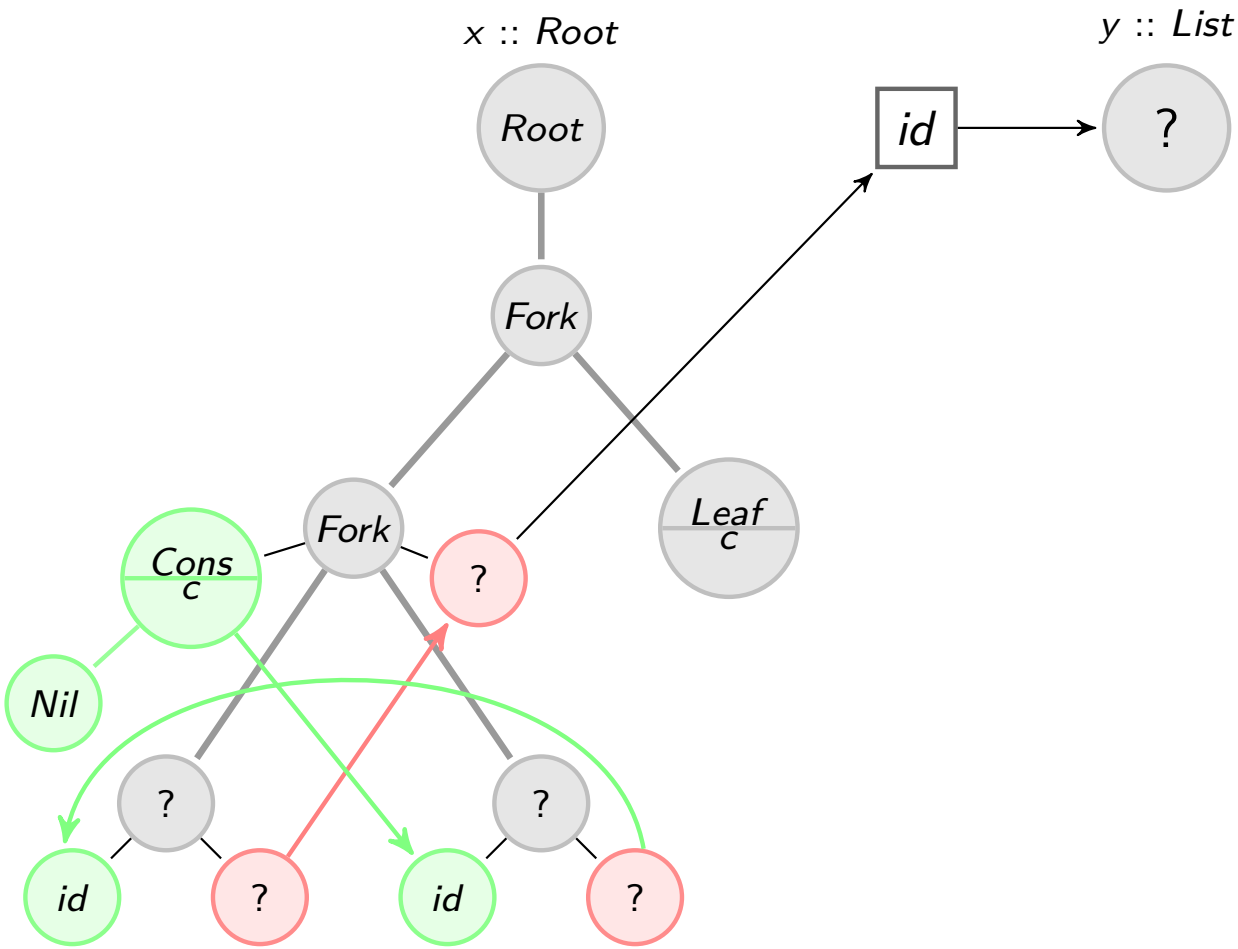
$$\begin{array}{lcl} y & = & x_1.s \\ x_1.h & = & Nil \end{array}$$

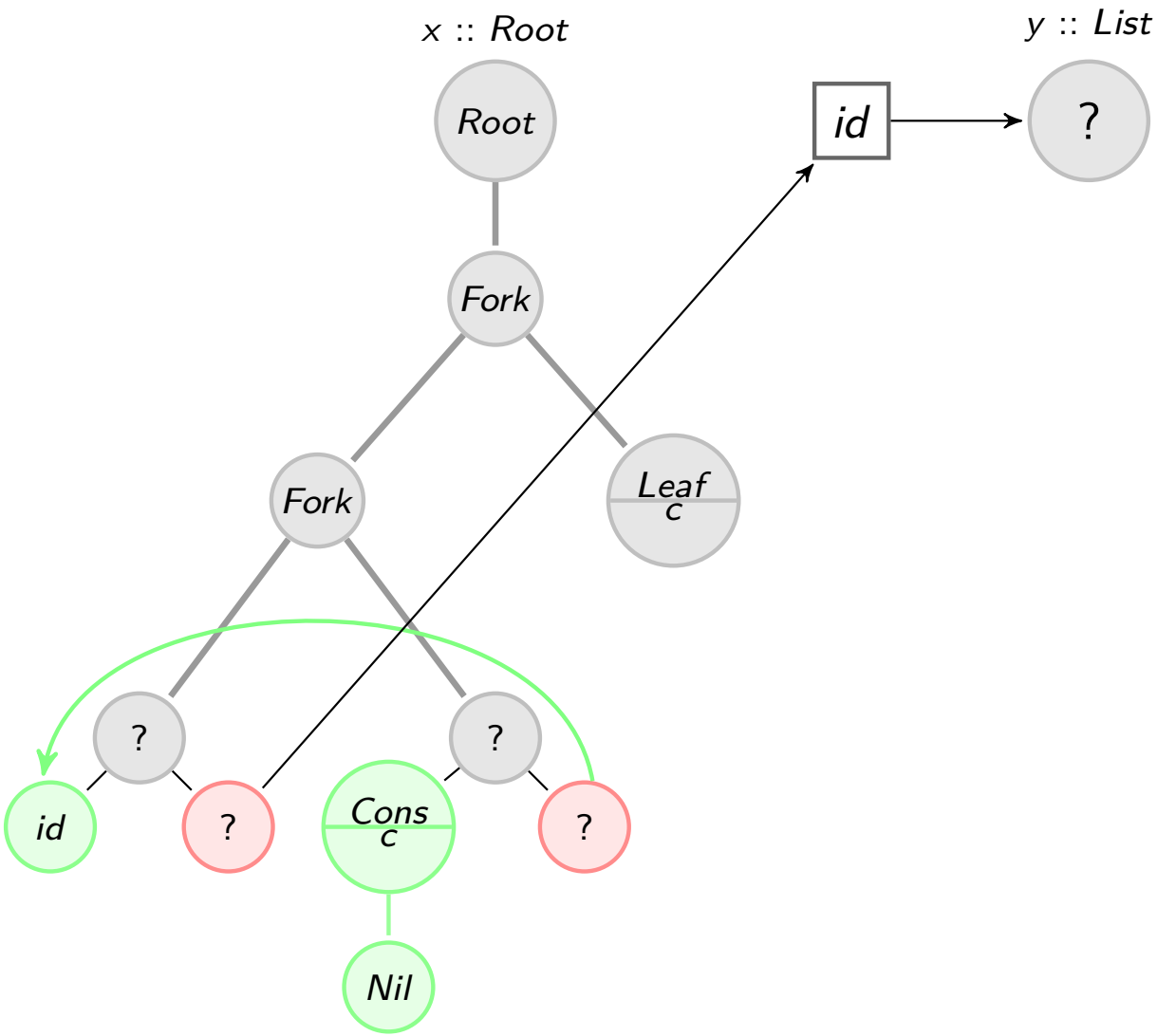

x :: Root
y :: List
Root
id
?
Nil
Fork
id
?
?
id
?
?
?

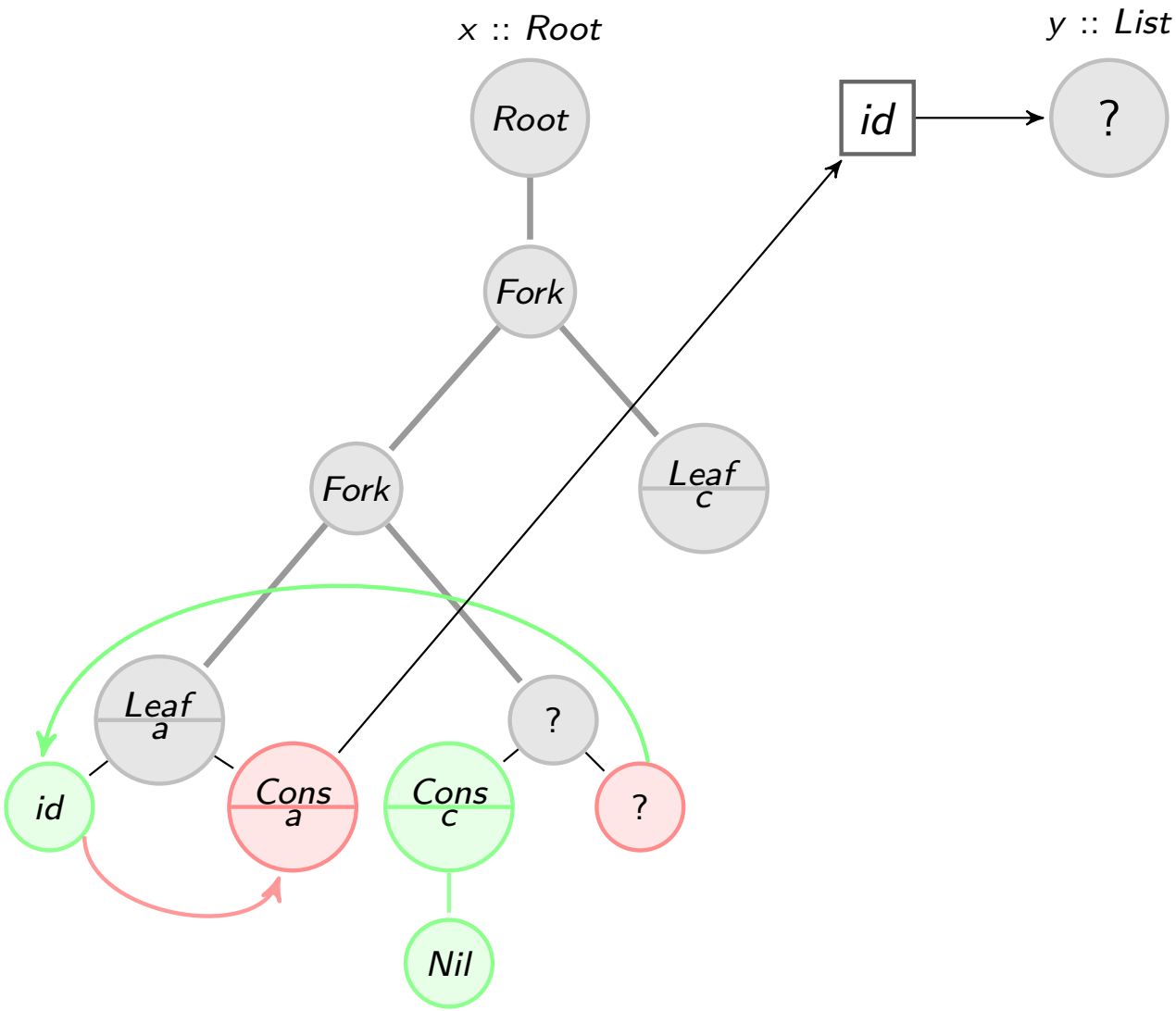

x :: Root
y :: List
Root
id
?
Fork
?
?
id
?
Nil
?

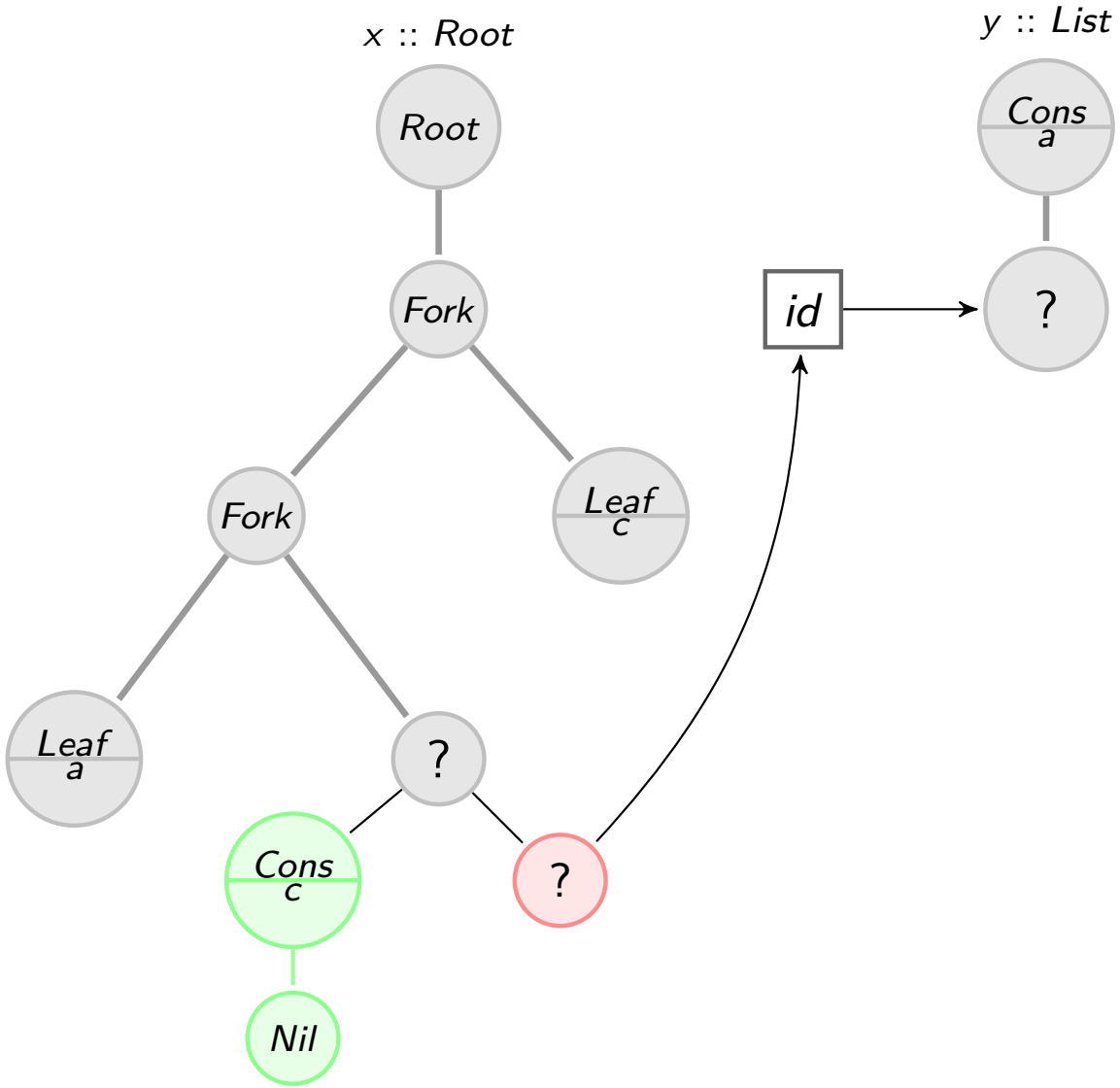

x :: Root
y :: List
Root
id
?
Fork
?
Leaf
c
id
?
Nil
Cons
c

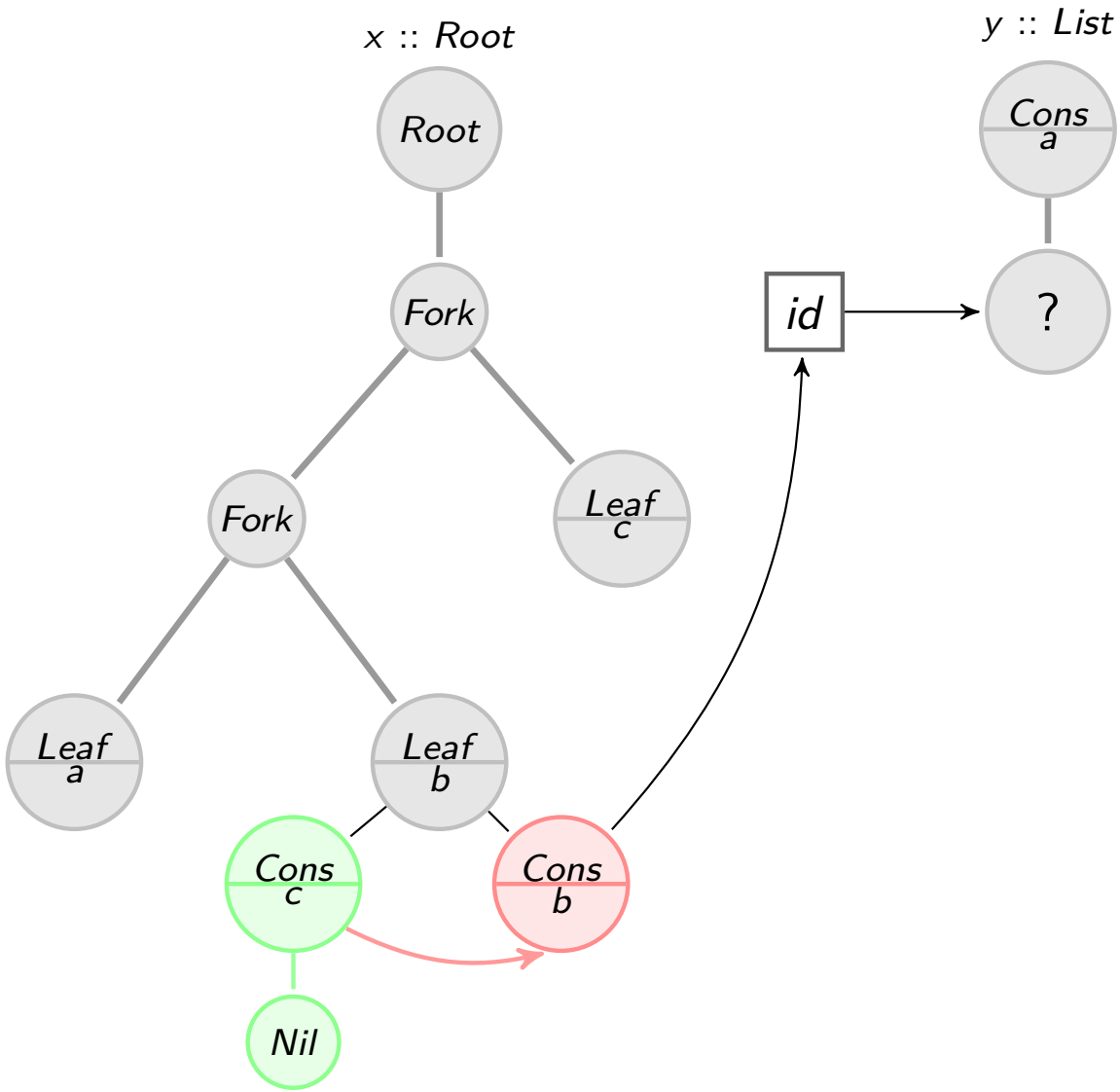

x :: Root
y :: List
Root
id
?
Fork
?
Leaf
c
Cons
c
?
Nil

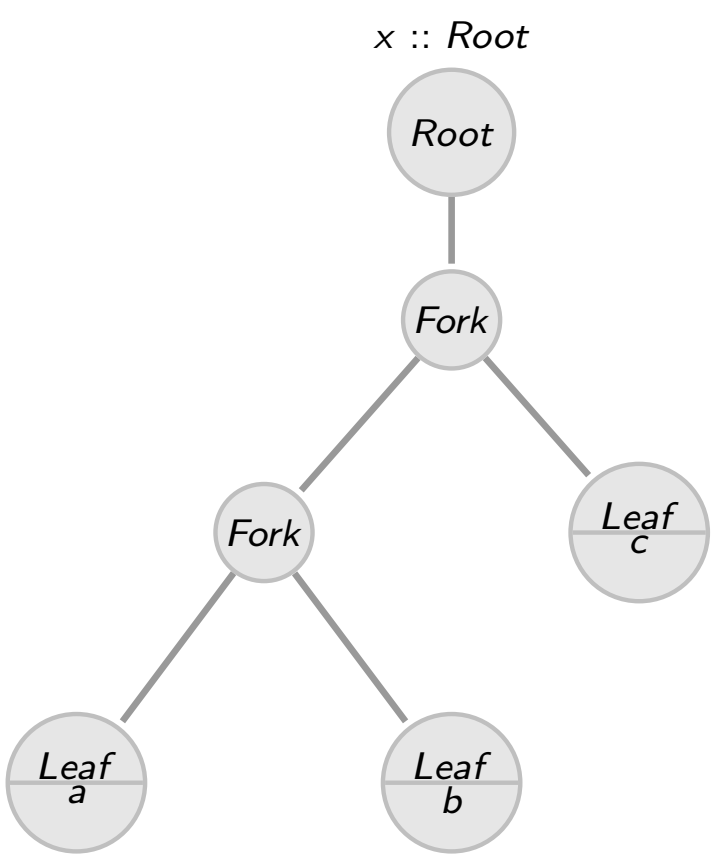

x :: Root
y :: List
Root
id
?
Fork
Fork
Leaf
c
Cons
c
?
Nil
?
?
id
?
id
?

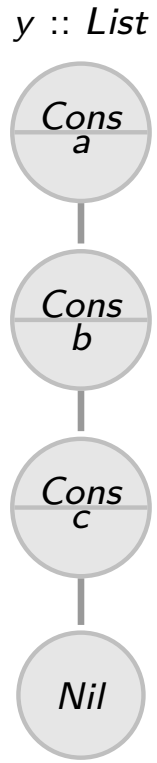

x :: Root
y :: List
Root
id
?
Fork
Fork
Leaf
c
?
?
id
?
Cons
c
?
Nil


x :: Root
y :: List
Root
id
?
Fork
Fork
Leaf
c
Leaf
a
?
id
Cons
a
Cons
c
?
Nil


x :: Root
y :: List
Root
Fork
Fork
Leaf
a
Leaf
c
?
Cons
c
?
Nil
id
Cons
a
?

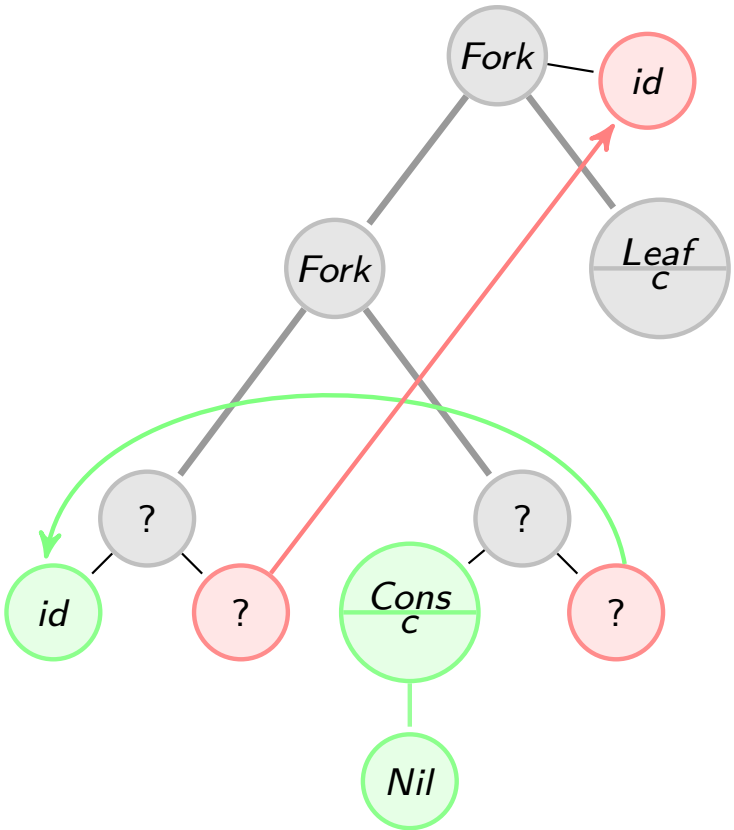

x :: Root
y :: List
Root
Fork
Fork
Leaf
c
Leaf
a
Leaf
b
Cons
c
Cons
b
Nil
id
Cons
a
?


x :: Root
Root
Fork
Fork
Leaf
c
Leaf
a
Leaf
b


y :: List
Cons
a
Cons
b
Cons
c
Nil

Composite productions

Derived operator $t^{@}$ for $t(x_1, x_2) = Fork(Fork(x_1, x_2), Leaf_c)$

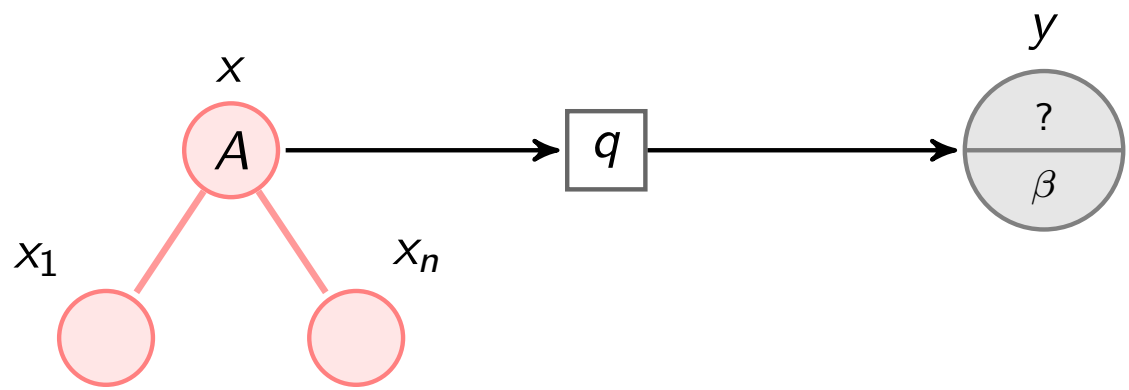


$$t^{@} : \langle x :: Bin \rangle \rightarrow \langle x_1 :: Bin \rangle \ \langle x_2 :: Bin \rangle$$

where
$$x \cdot s = x_1 \cdot s$$
$$x_1 \cdot h = x_2 \cdot s$$
$$x_2 \cdot h = Cons_a(Nil)$$

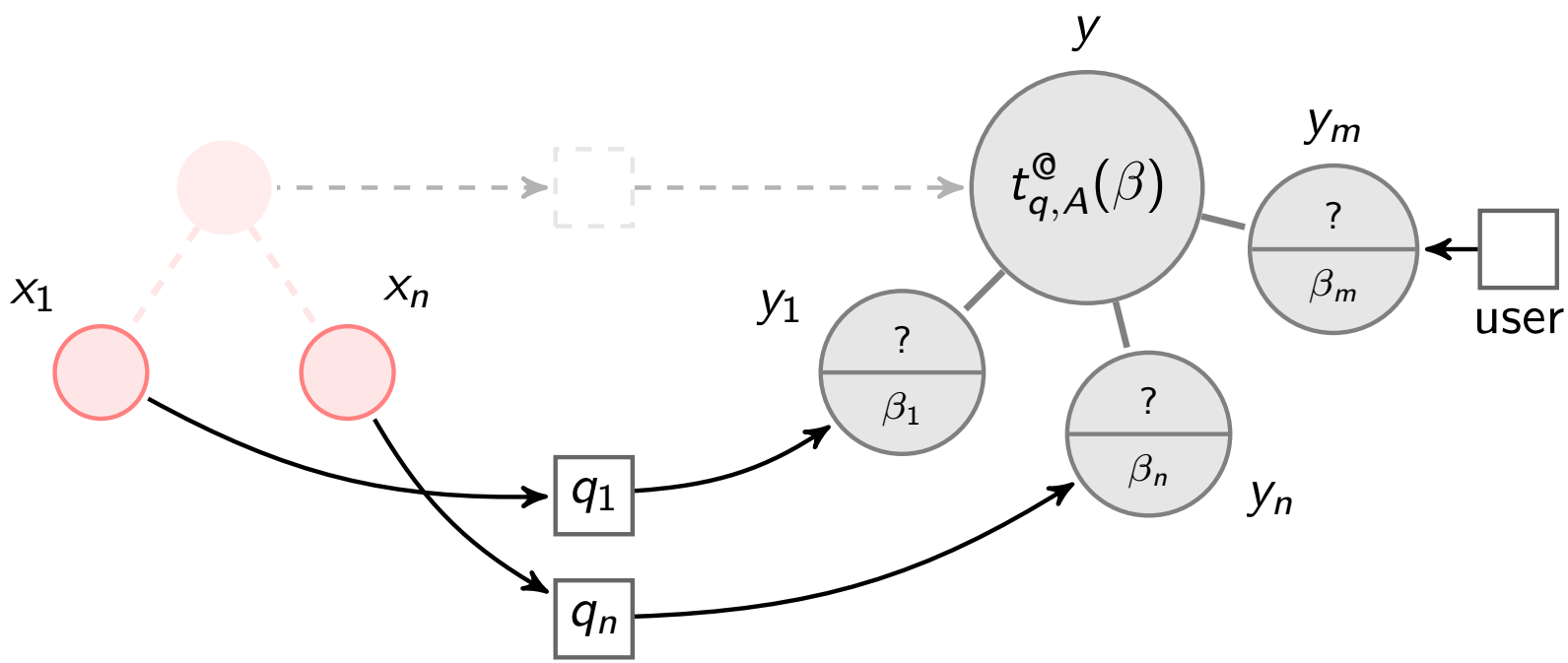
Generators (tree transducers)

$q(A(x_1, \cdots, x_n)) = \cdots$



Generators (tree transducers)

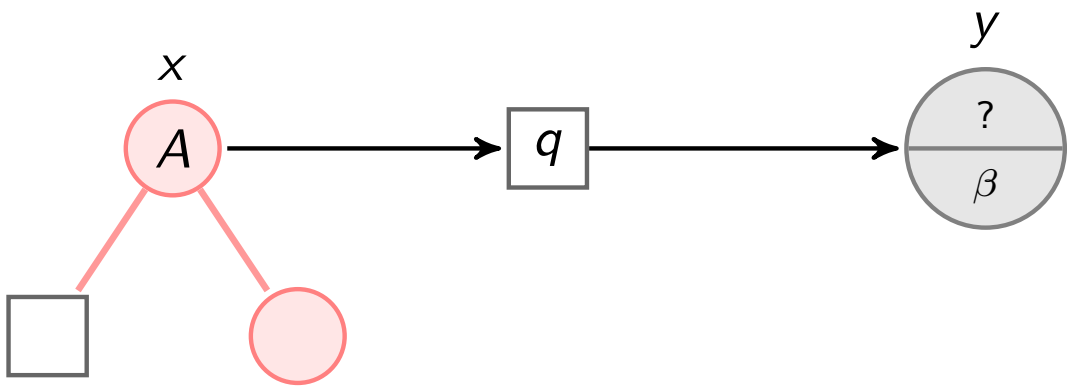
$q(A(x_1, \cdots, x_n)) = t_{q,A}(\beta)(y_1, \cdots, y_m)$
where $y_1 = q_1(x_{i_1}), \ldots, y_n = q_n(x_{i_n})$



Adding handles in semantic rules

The handle of a generator can be used in a semantic rule in order to linked it to an input place. With some restriction: Any attribute whose value may contain an handle should be used only once in semantics rules (a generator should never be linked to multiple input places).

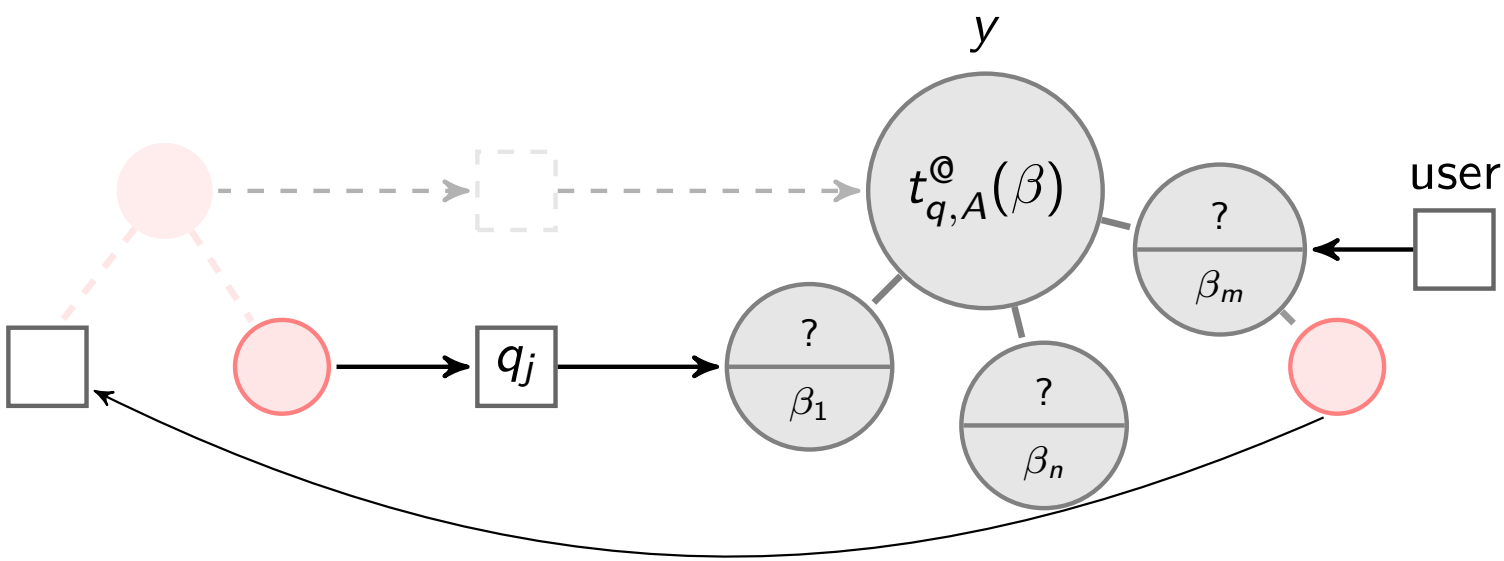
$q(A(x_1, \cdots, x_n)) = \cdots$



Adding handles in semantic rules

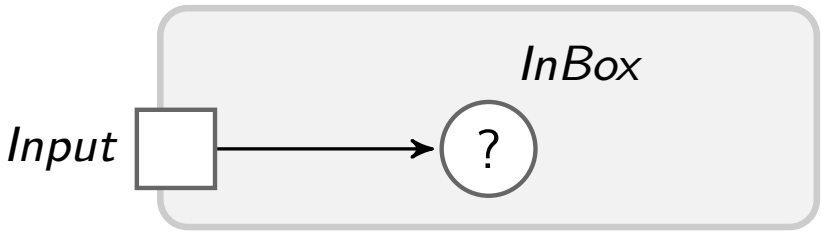
The handle of a generator can be used in a semantic rule in order to linked it to an input place. With some restriction: Any attribute whose value may contain an handle should be used only once in semantics rules (a generator should never be linked to multiple input places).

$$q(A(x_1, \cdots, x_n) = t_{q,A}(\beta)(y_1, \cdots, y_m)) \textbf{ where } \begin{cases} y_j & = & q_j(x_{i_j}) \\ y_{j'} & = & user \\ x_i & = & a \end{cases}$$

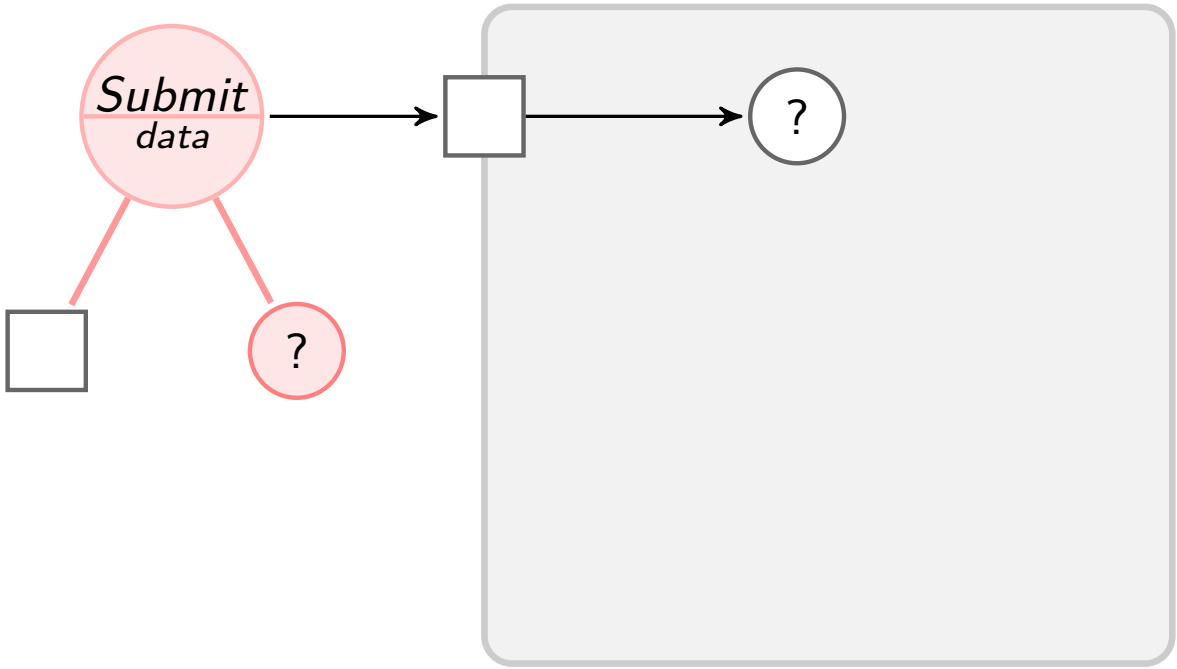


Communication between stakeholders: Public channels

Each site contains some input channels (each connected to an InBox). Upon receiving a data they behave according to the schema described above,

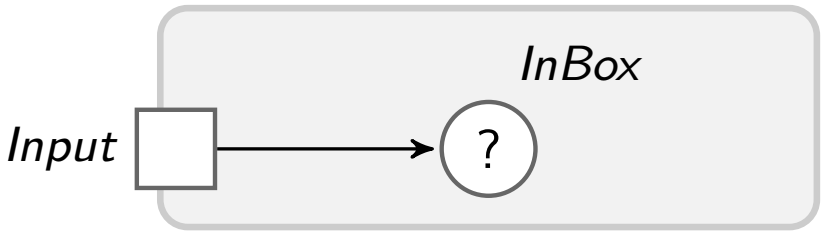


except that this channel does not disappear but is "redirected" to a new fresh cell of the InBox (implemented as a list of the artifacts created by the messages)

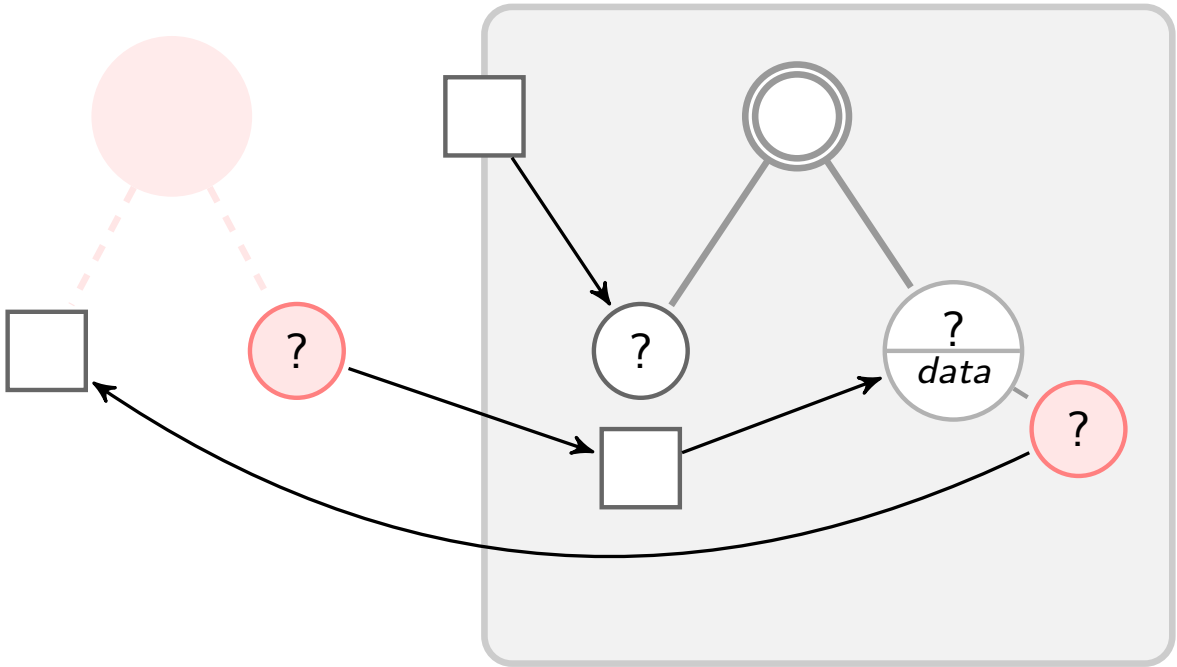


Communication between stakeholders: Public channels

Each site contains some input channels (each connected to an InBox). Upon receiving a data they behave according to the schema described above,



except that this channel does not disappear but is "redirected" to a new fresh cell of the InBox (implemented as a list of the artifacts created by the messages)



Assessment of the Model

and Future Works

in collaboration with
Héla Gomri, Georges-Edouard Kouamou, Célestin Nkuimi,
Rodrigue Djeumen, ...

Language-oriented approach

Language-oriented approach Procedural parts of artifacts encode and encapsulate technical known-hows that end user may safely ignore. Each stakeholder manipulates documents in a familiar syntax using notations adapted to his domain (Domain Specific Languages).

Language-oriented approach

Language-oriented approach Procedural parts of artifacts encode and encapsulate technical known-hows that end user may safely ignore. Each stakeholder manipulates documents in a familiar syntax using notations adapted to his domain (Domain Specific Languages).

Configuration of the Working Environment Attribute grammars can be used to design DSL as a set of functional combinators. (Swierstra et al).

Language-oriented approach

Language-oriented approach Procedural parts of artifacts encode and encapsulate technical known-hows that end user may safely ignore. Each stakeholder manipulates documents in a familiar syntax using notations adapted to his domain (Domain Specific Languages).

Configuration of the Working Environment Attribute grammars can be used to design DSL as a set of functional combinators. (Swierstra et al).

Component-based Architecture Bottom-up approach: we first design the notations specific to each stakeholder then we write specific procedures using these DSLs. Therefore related applications can share the same architecture (product line approach vs a particular application).

Language-oriented approach

Language-oriented approach Procedural parts of artifacts encode and encapsulate technical known-hows that end user may safely ignore. Each stakeholder manipulates documents in a familiar syntax using notations adapted to his domain (Domain Specific Languages).

Configuration of the Working Environment Attribute grammars can be used to design DSL as a set of functional combinators. (Swierstra et al).

Component-based Architecture Bottom-up approach: we first design the notations specific to each stakeholder then we write specific procedures using these DSLs. Therefore related applications can share the same architecture (product line approach vs a particular application).

Combinator-based communication Component-based programming relies on an adaquate notion of interface for the communication between the various entities. To communicate with an external service, we use (part of) its language. Interfaces consists in the type (and informal semantical description) of combinators: each DSL is embedded in the same host language (e.g. Haskell) so that interfacing the various activities does not involve extra machinery.

User-centric Approach

Collaborative Systems The system is no longer dedicated to the orchestration of the various activities but it is rather design as a support to the structuration of its own activity and to the communication with the external services. Exchanged documents (active forms) appears more as a support to the communication than as the result of this collaboration.

User-centric Approach

Collaborative Systems The system is no longer dedicated to the orchestration of the various activities but it is rather design as a support to the structuration of its own activity and to the communication with the external services. Exchanged documents (active forms) appears more as a support to the communication than as the result of this collaboration.

Service-oriented Architecture Some combinators provided in the interface of a component may implement services. A call to a service can be presented as a form (build from these combinators) that enacts queries and specific side effect actions (sending a message, updating a data base ...).

User-centric Approach

Collaborative Systems The system is no longer dedicated to the orchestration of the various activities but it is rather design as a support to the structuration of its own activity and to the communication with the external services. Exchanged documents (active forms) appears more as a support to the communication than as the result of this collaboration.

Service-oriented Architecture Some combinators provided in the interface of a component may implement services. A call to a service can be presented as a form (build from these combinators) that enacts queries and specific side effect actions (sending a message, updating a data base ...).

Web Technology The approach is declarative, entities communicates through asynchronous message passing (without fifo assumption nor share memory), and everthing is encoded as (active) structured documents (local states=active folders, message contents=active forms). These systems may thus be directly deployed on Internet and in particular in a degraded environement (where a client/server approach would not be convenient).

Modularity

The attribute grammars approach to the design of combinator languages allows various forms of modularity:

- modular attribute grammars based on Bekic principle of resolution of systems of recursive equation by substitution allows to decompose a large attribute grammar into a set of smaller specifications. This method encourages the reuse of langage designs.

Modularity

The attribute grammars approach to the design of combinator languages allows various forms of modularity:

- modular attribute grammars based on Bekic principle of resolution of systems of recursive equation by substitution allows to decompose a large attribute grammar into a set of smaller specifications. This method encourages the reuse of langage designs.
- Descriptional composition A way of decomposing a langage into successive stages (originally introduced by Genzinger and Giegerich). Allow semantic rules to use notations borrowed from other pre-existing languages.

Modularity

The attribute grammars approach to the design of combinator languages allows various forms of modularity:

- modular attribute grammars based on Bekic principle of resolution of systems of recursive equation by substitution allows to decompose a large attribute grammar into a set of smaller specifications. This method encourages the reuse of langage designs.
- Descriptional composition A way of decomposing a langage into successive stages (originally introduced by Genzinger and Giegerich). Allow semantic rules to use notations borrowed from other pre-existing languages.
- Aspect decomposition An aspect can be viewed as a bunch of attributes.

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies
  1. The editorial process of an electronic journal (from submission to publication).

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies
  1. The editorial process of an electronic journal (from submission to publication).
  2. A (simplified) system of distance learning in a degraded environment (intermitent connection to Internet).

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies
  1. The editorial process of an electronic journal (from submission to publication).
  2. A (simplified) system of distance learning in a degraded environment (intermitent connection to Internet).
  3. A distributed system of early detection of diseases (Institut Pasteur).

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies
  1. The editorial process of an electronic journal (from submission to publication).
  2. A (simplified) system of distance learning in a degraded environment (intermitent connection to Internet).
  3. A distributed system of early detection of diseases (Institut Pasteur).
  4. A reporting system ("write things once" principle, 90% of an activity report consists of information already available, avoid unnecessary overload in low-content communication: "zero Email" principle).

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies
  1. The editorial process of an electronic journal (from submission to publication).
  2. A (simplified) system of distance learning in a degraded environment (intermitent connection to Internet).
  3. A distributed system of early detection of diseases (Institut Pasteur).
  4. A reporting system ("write things once" principle, 90% of an activity report consists of information already available, avoid unnecessary overload in low-content communication: "zero Email" principle).
- An editor for the description and generation of a document-centric worflow system based on Active Folders. Using the early version of an editor of modular attribute grammars that we have developped (Edigram).

Future Works

- Introduce notations and diagrams to guide the modelisation process (as an aid to the design of particular instances of Active Folders).
- Automatic generation of code from the such a specification.
- We would like to develop some representative case studies
  1. The editorial process of an electronic journal (from submission to publication).
  2. A (simplified) system of distance learning in a degraded environment (intermitent connection to Internet).
  3. A distributed system of early detection of diseases (Institut Pasteur).
  4. A reporting system ("write things once" principle, 90% of an activity report consists of information already available, avoid unnecessary overload in low-content communication: "zero Email" principle).
- An editor for the description and generation of a document-centric worflow system based on Active Folders. Using the early version of an editor of modular attribute grammars that we have developped (Edigram).
- Formal properties: modularity, interfaces, soundness ...