Optimization of Visitor Performance by Reflection-Based Analysis

Markus Lepper[1] Baltasar Trancón y Widemann [1 2]

[1] <semantics/> GmbH, Berlin post@markuslepper.eu

[2] Universität Bayreuth Baltasar.Trancon@uni-bayreuth.de

Zürich, ICMT2011, 28. June 2011

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything which is inside a computer
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything which is inside a computer
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything which is inside a computer
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything which is inside a computer
  is a model.
  Every computation is a model transformation.

Visitors

- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

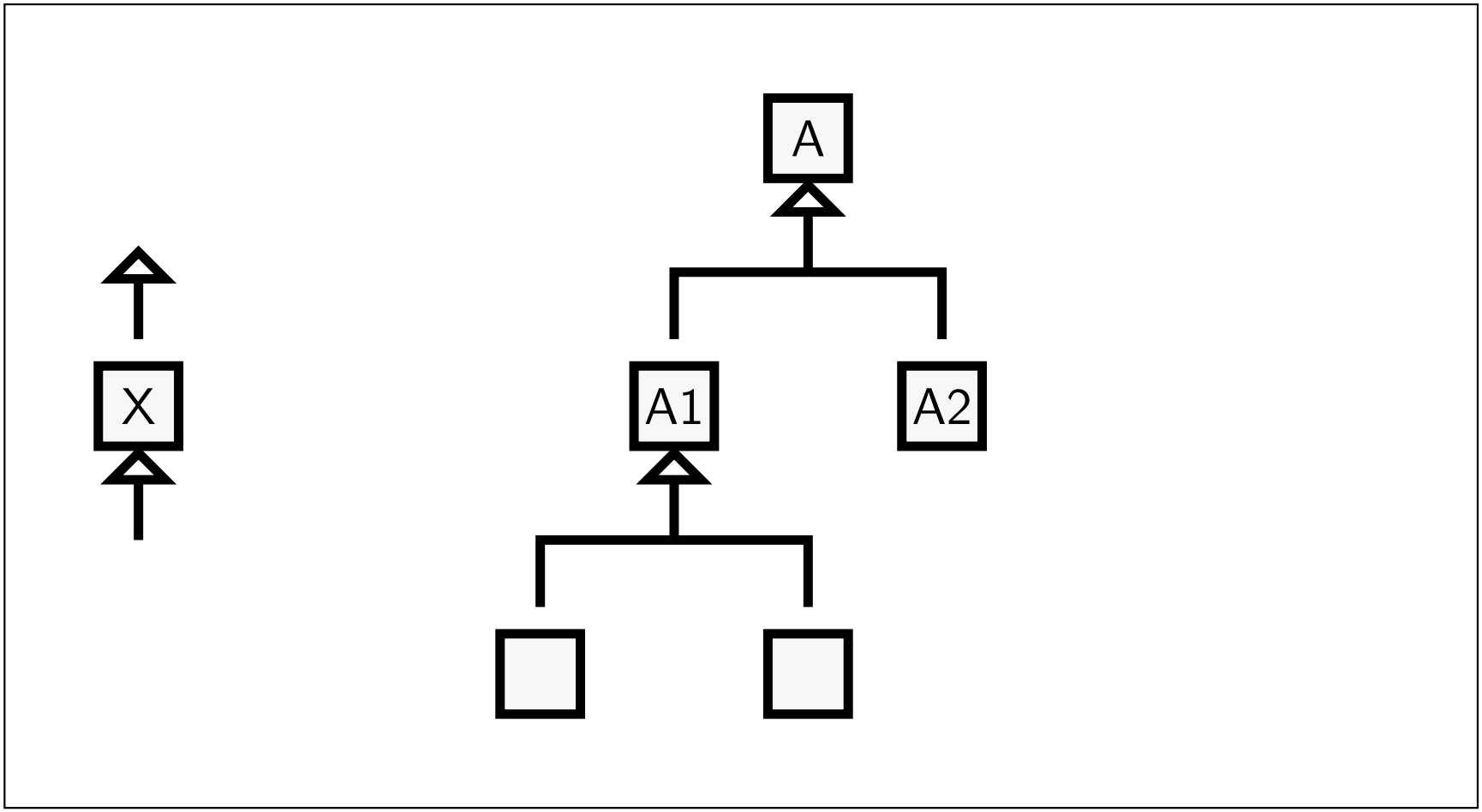
Authors in General

- Compiler Construction / Language Design
- OPAL / DSLs / Specification Languages (TTCN-3, TCI, Z) / Temporal Logics / XML / Signal Processing
- $^{\text{meta}}$–tools — Collection of all our tools for generic programming, compiler construction and language processing
- Everything which is inside a computer is a model.
  Every computation is a model transformation.

Visitors

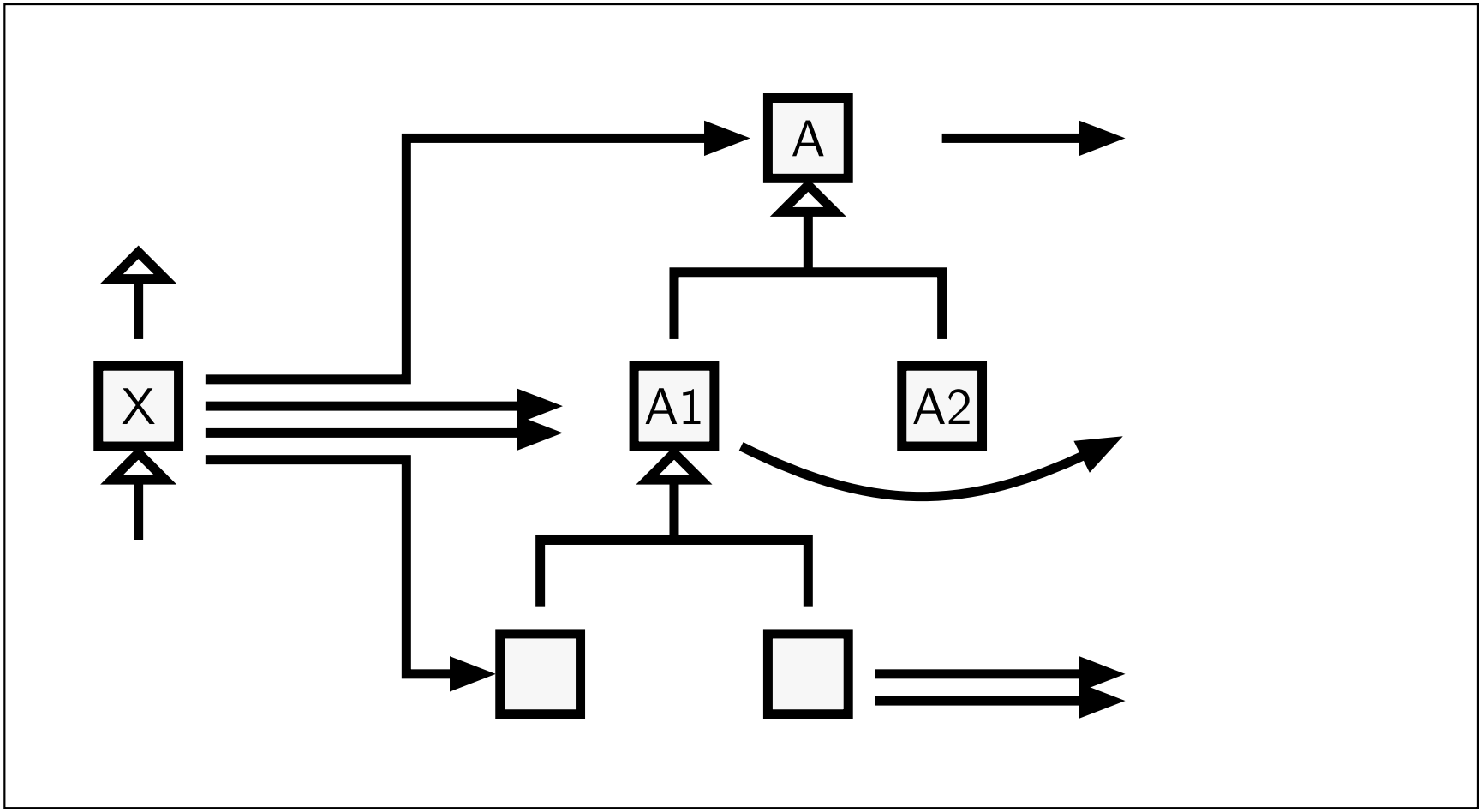
- Well-known design pattern (source: folklore)
- Combine imperative and declarative features
- E.g. separation of semantics / local code vs. global flow control
- Certain *robustness* against changes of the model structure

A Model is a collection of instances of classes

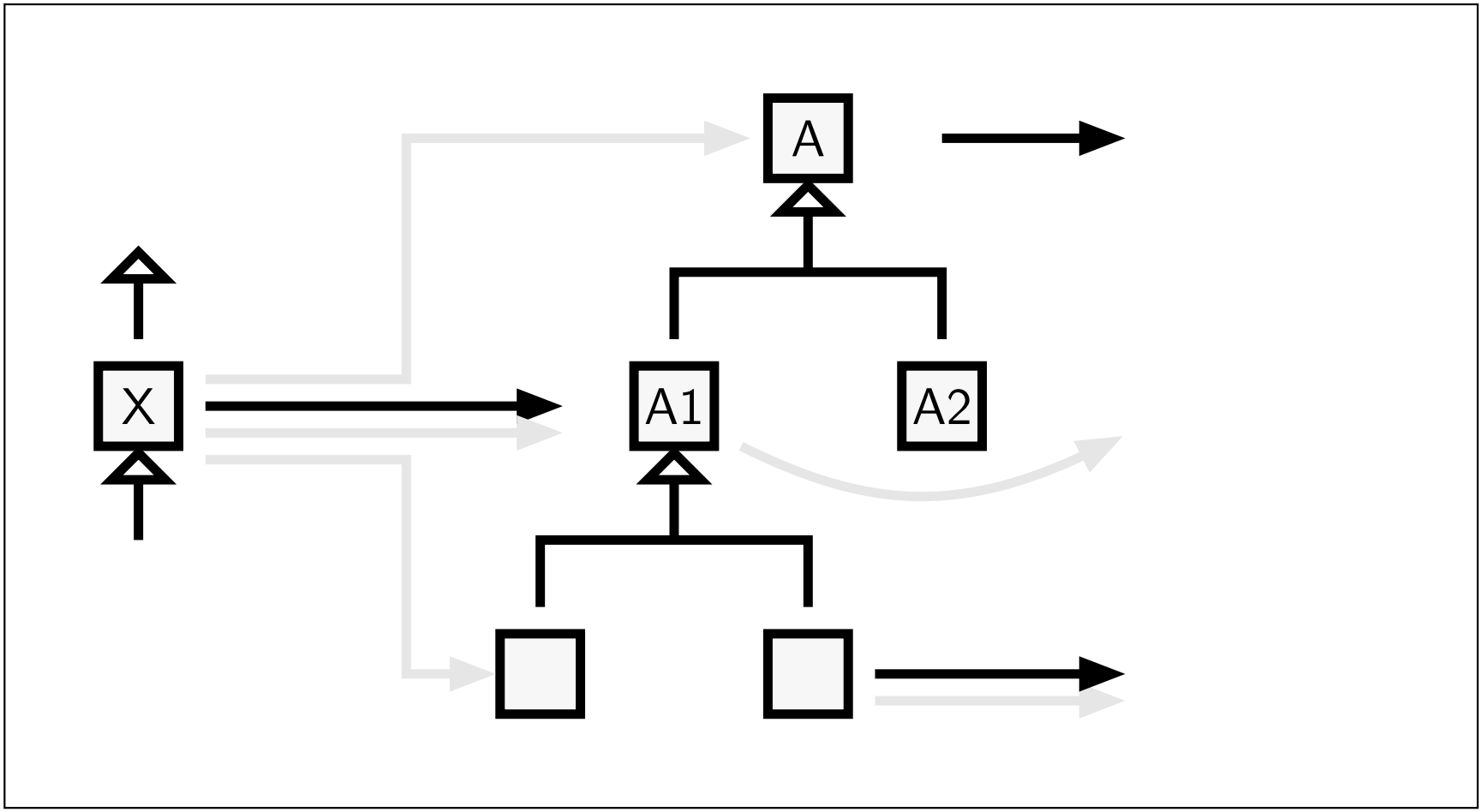


A Model is a collection of instances of classes

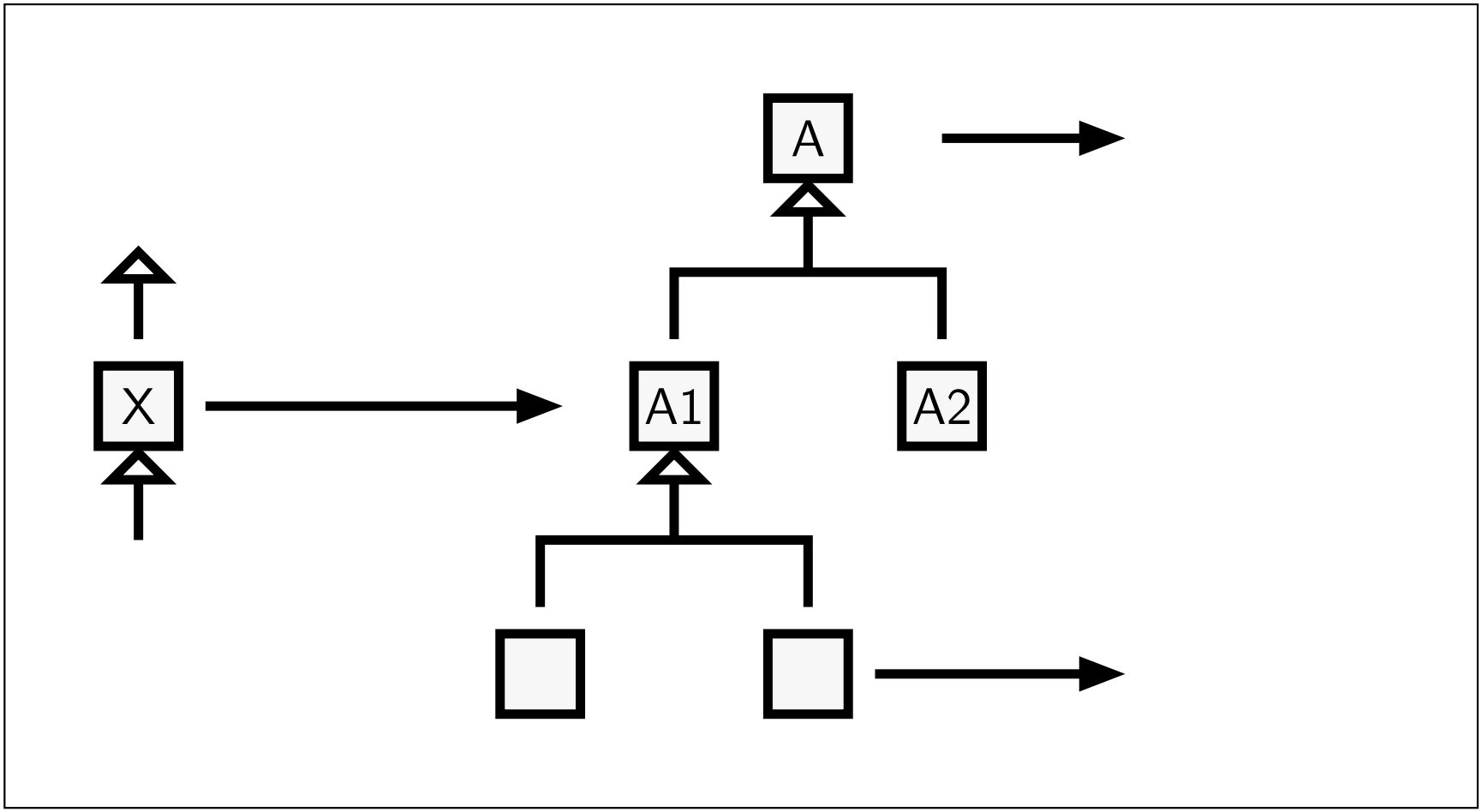
. . . with associations between instances



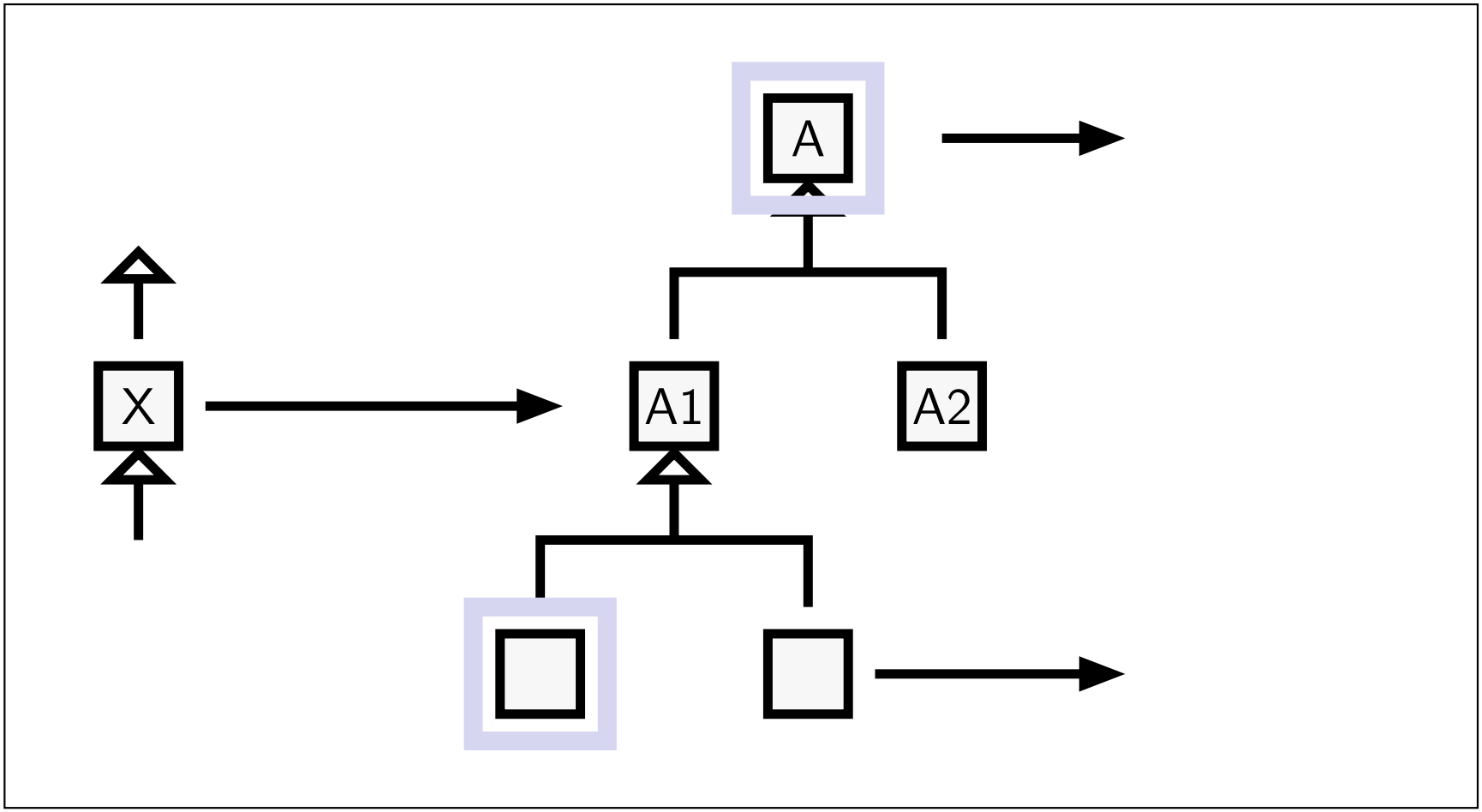
A Model is a collection of instances of classes
. . . with selected associations, defining visiting policy



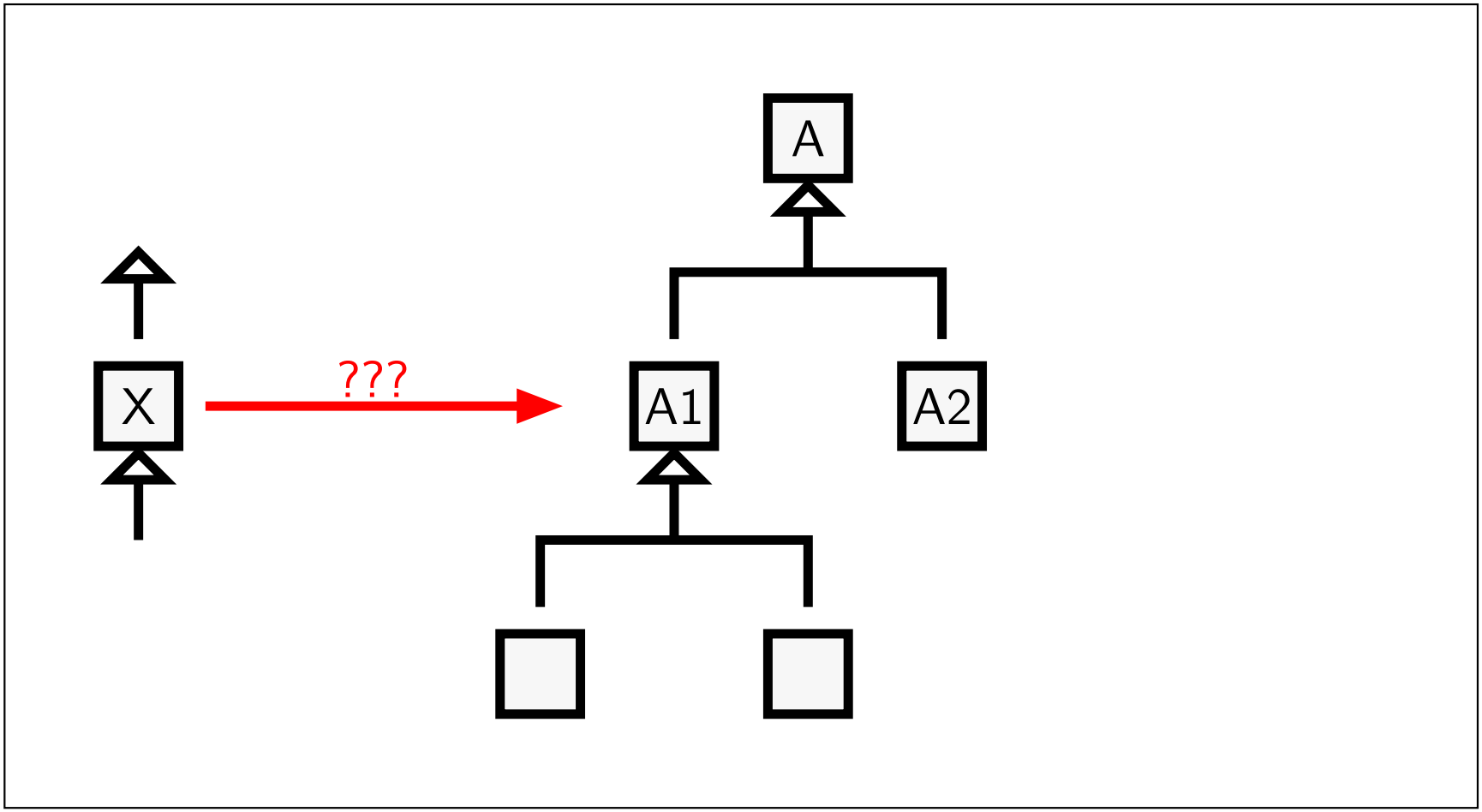
Base visitor: Mere traversal code
(possibly generated automatically)



Base visitor: Mere traversal code
Derived visitor: userdef'd semantic actions

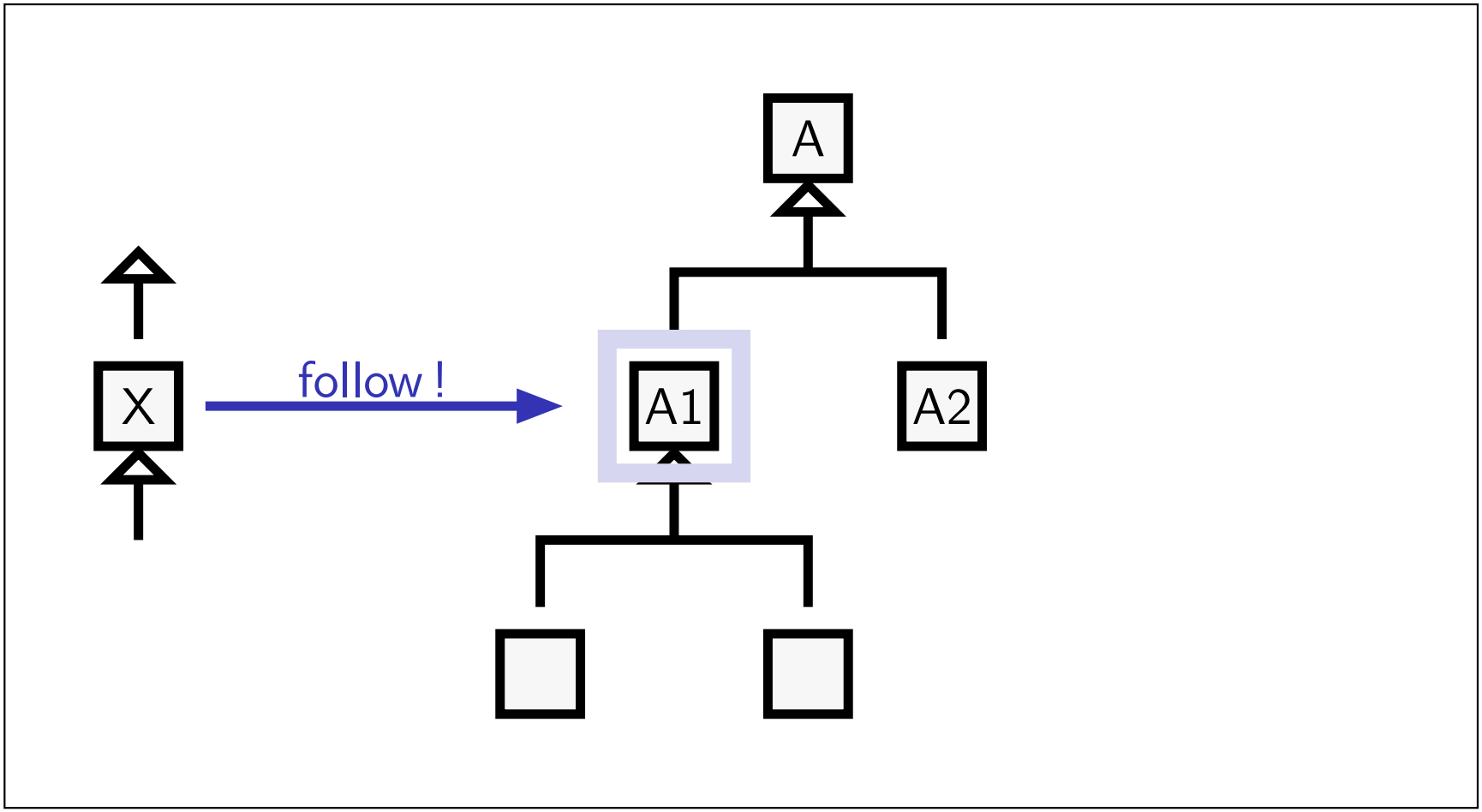


Is it necessary to follow a certain association?



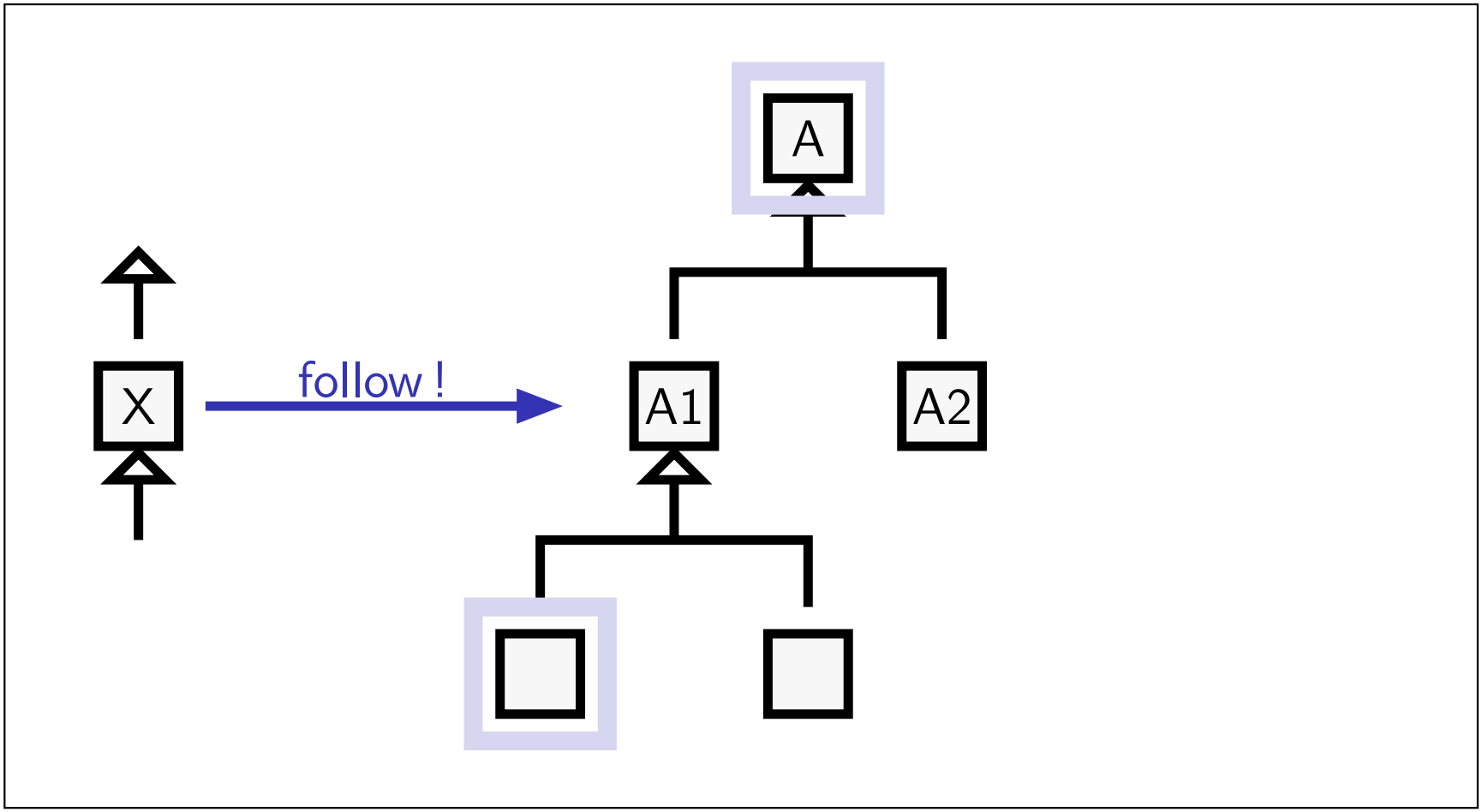
Is it necessary to follow a certain association?

. . . Yes, whenever user-defined code is directly reachable



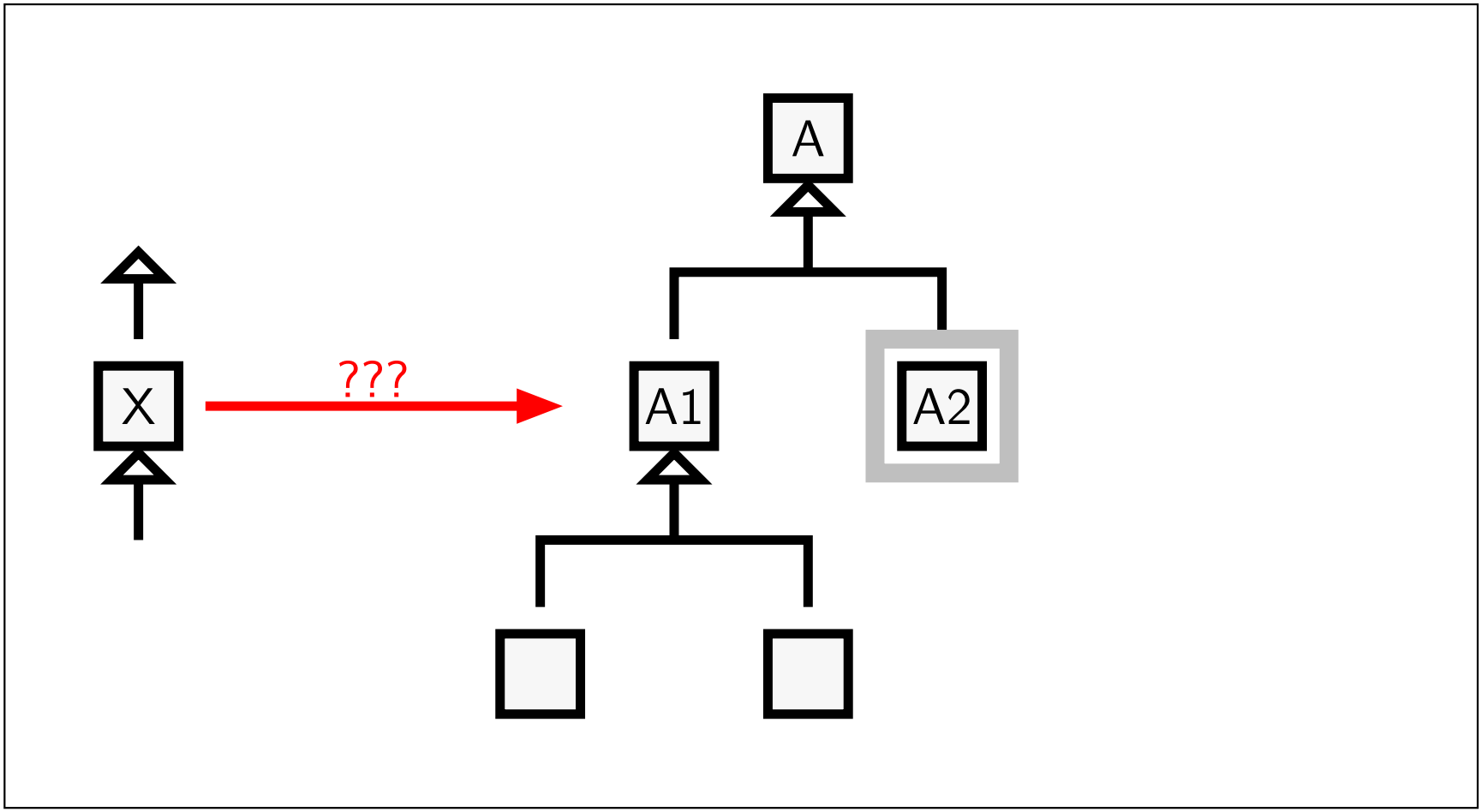
Is it necessary to follow a certain association?

. . . Yes, user-defined code reachable by inheritance/specialization



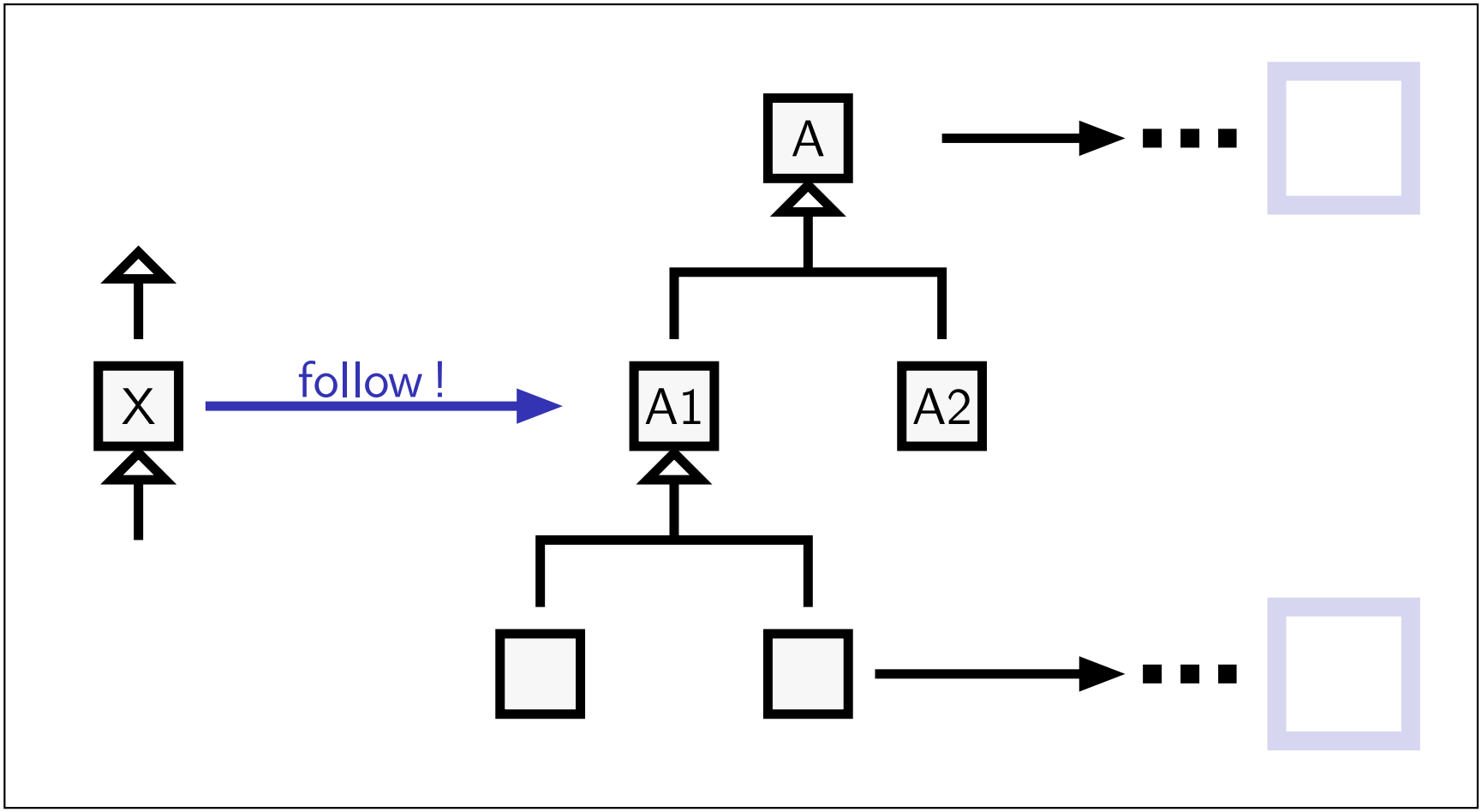
Is it necessary to follow a certain association?

. . . No, for nieces and nephews!

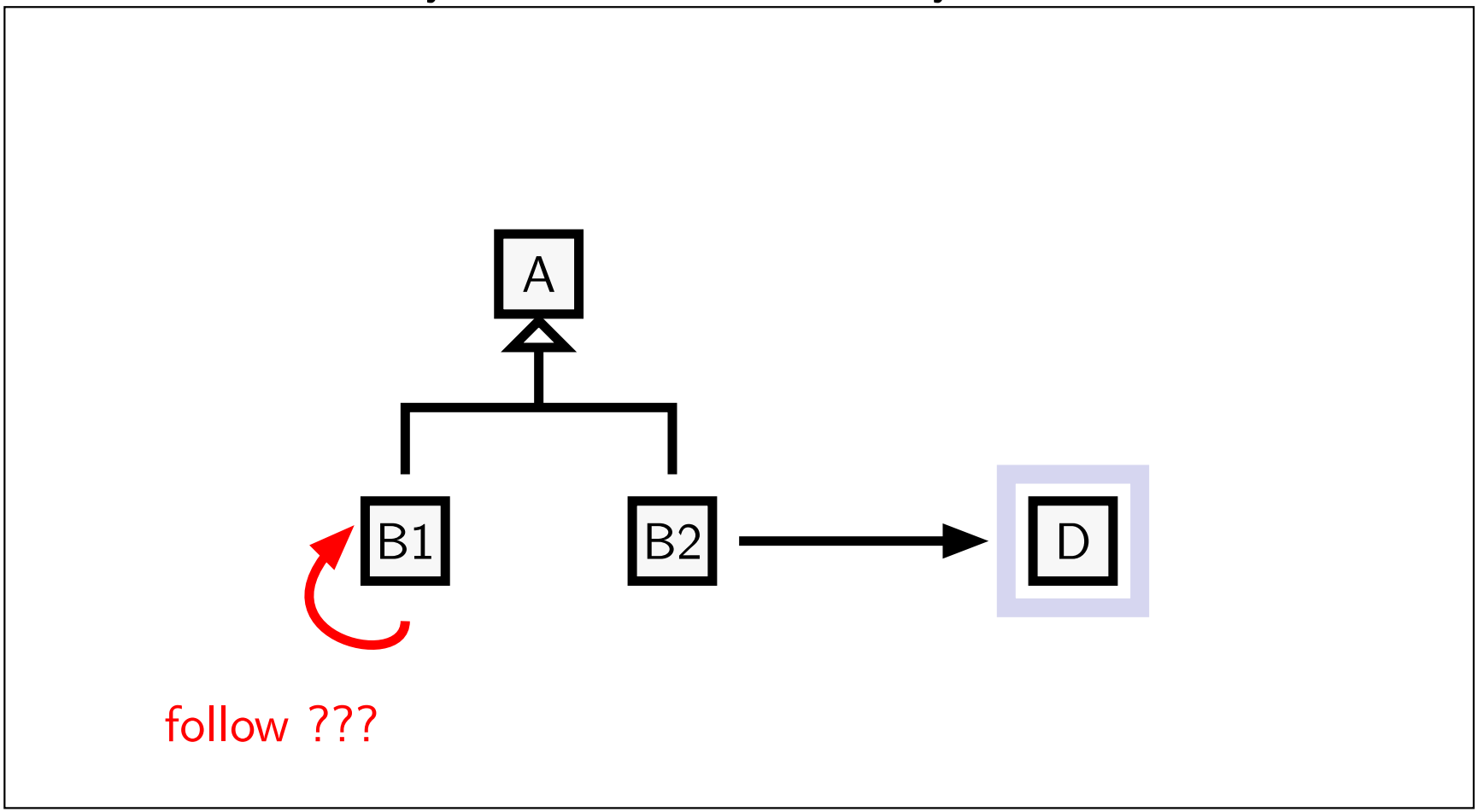


Is it necessary to follow a certain association?

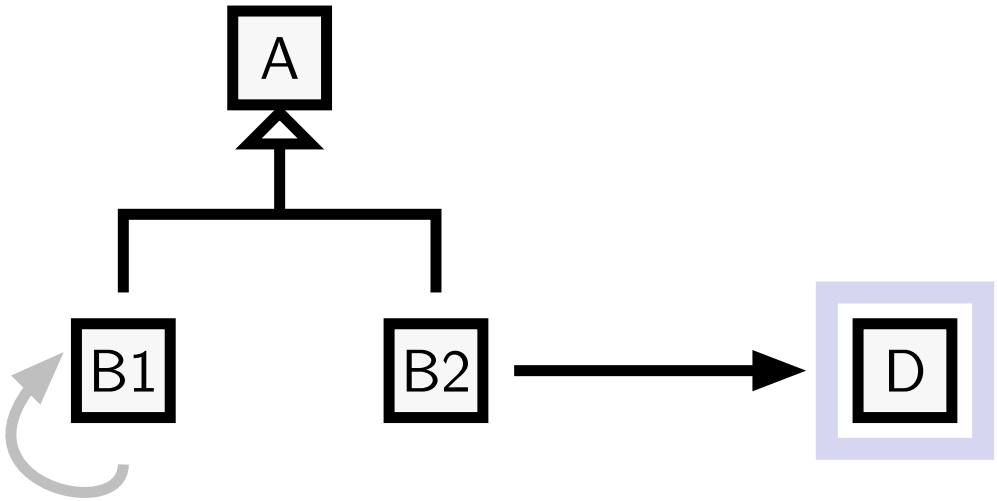
. . . Yes, when reaching user-defined code *indirectly*!



Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?

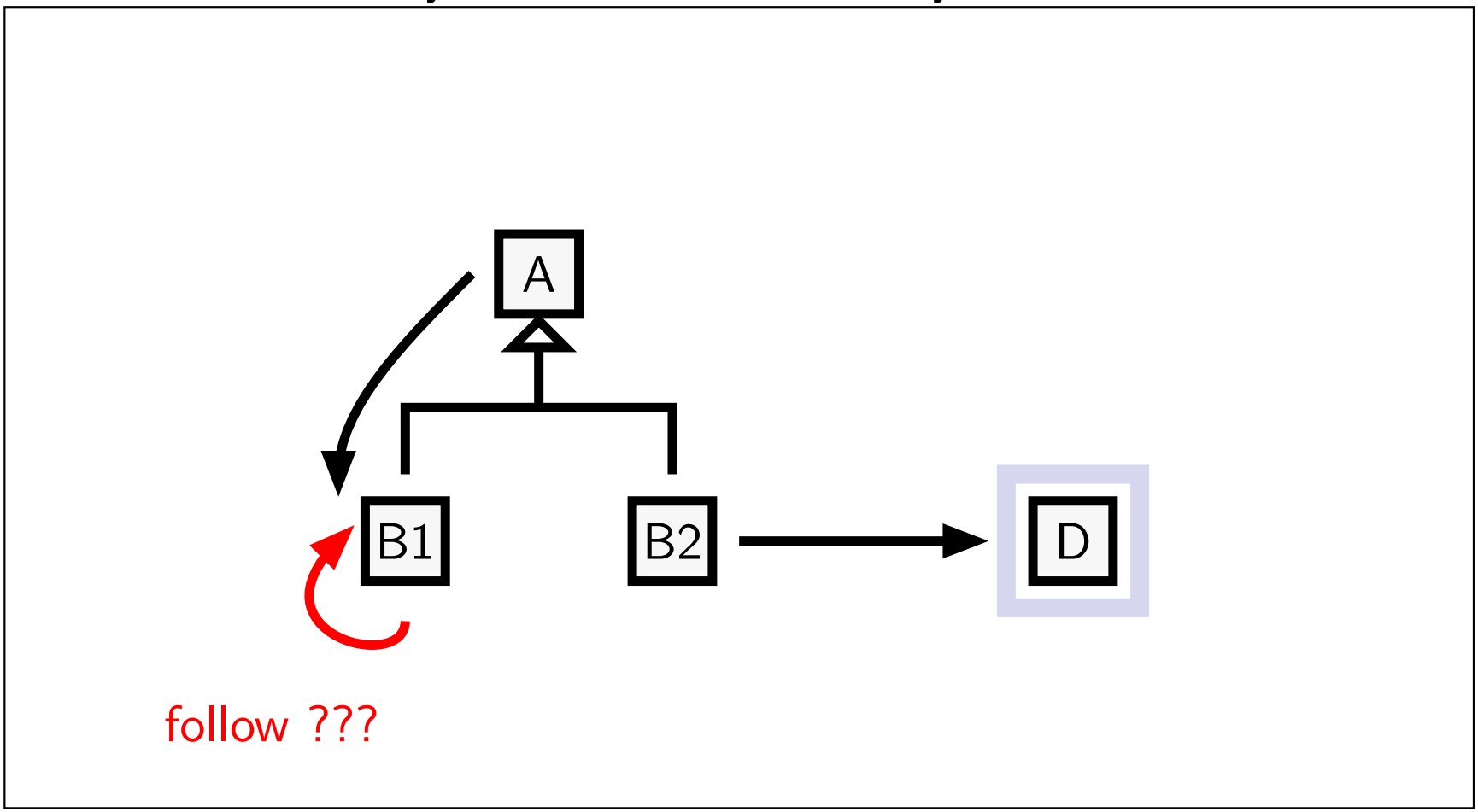


Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?

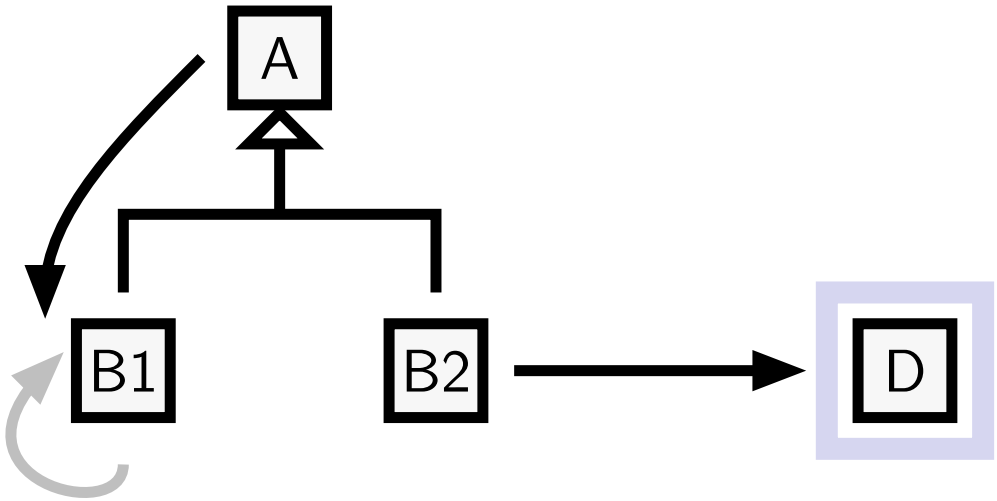


do NOT follow!

Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?

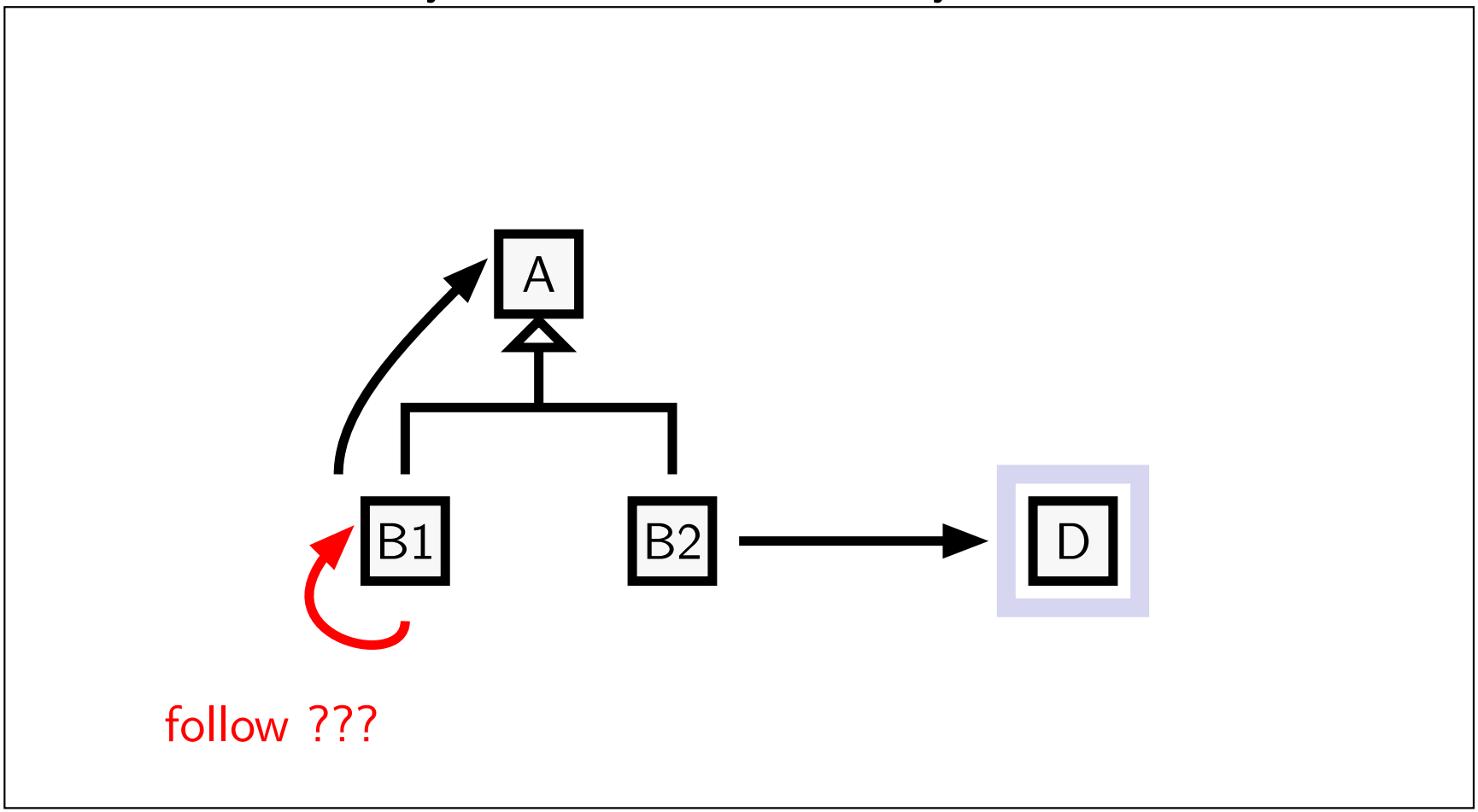


Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?

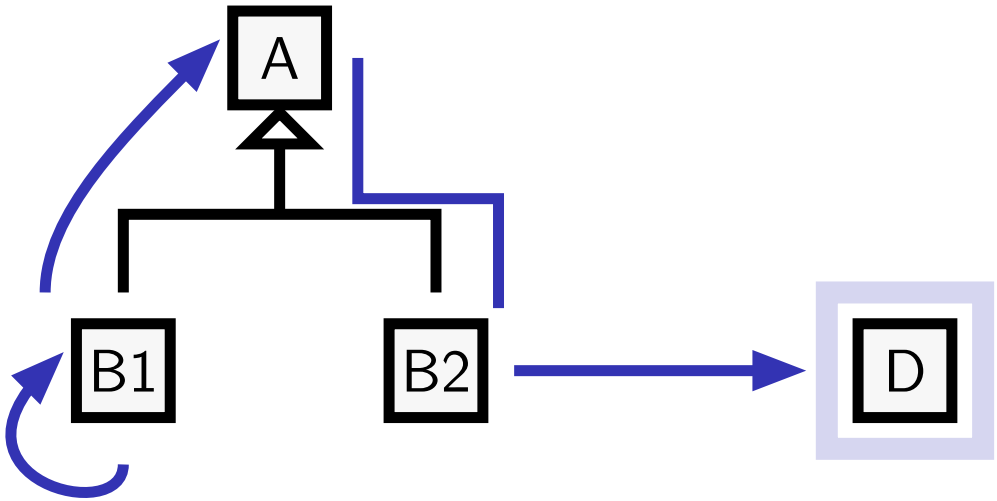


do NOT follow!

Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?

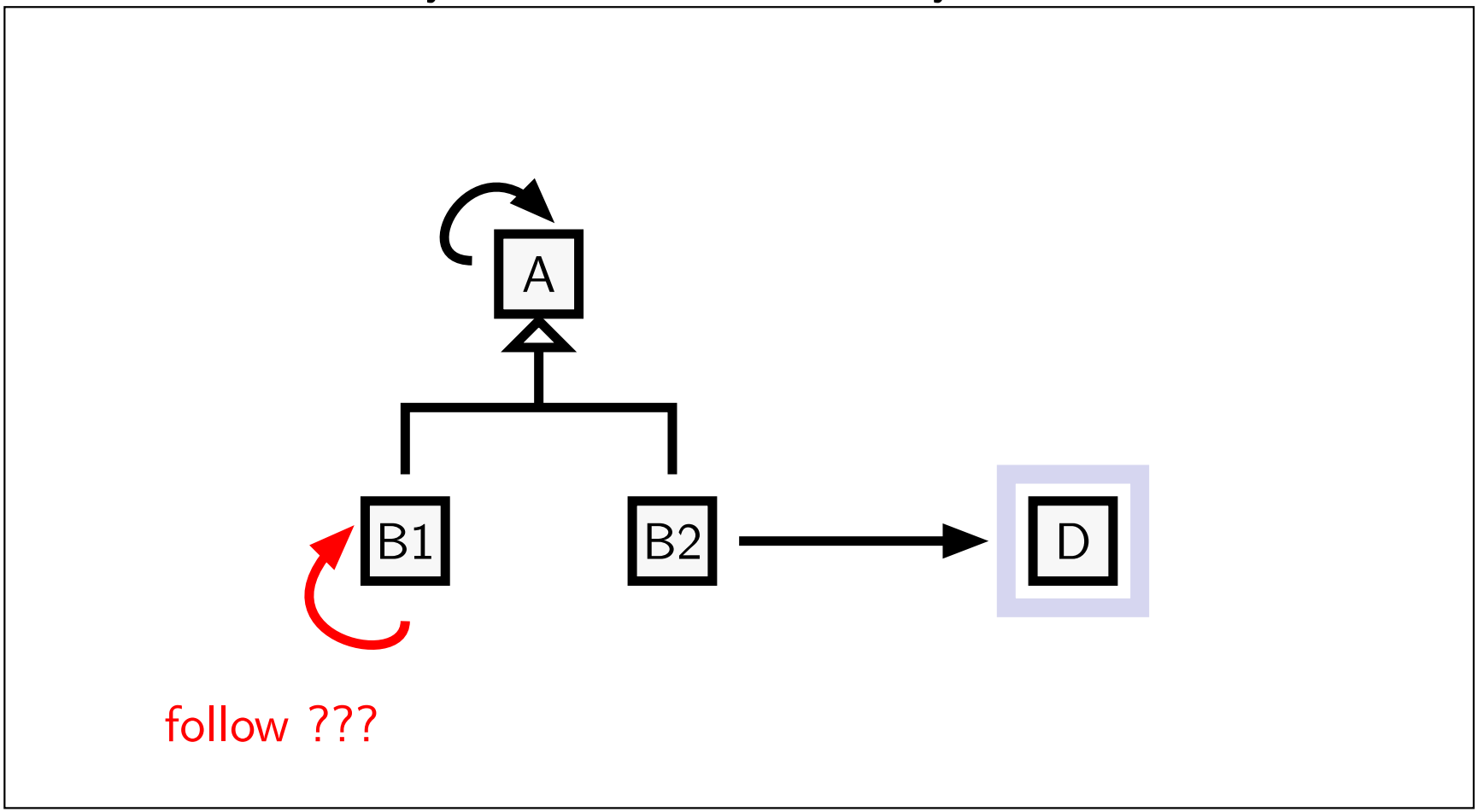


Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?

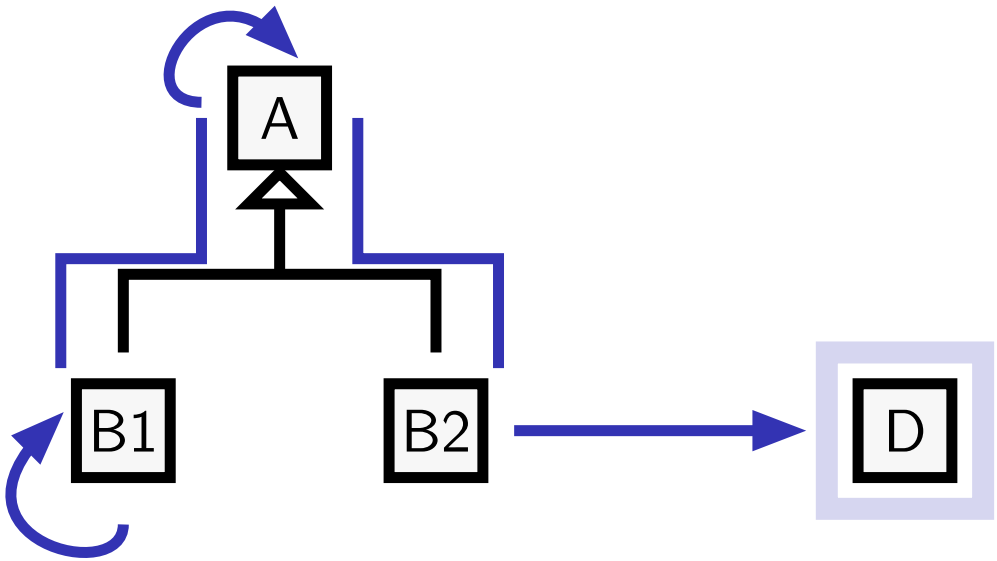


DO follow!

Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?



Is it necessary to follow a certain association?
User def'd code only for D. — Follow B1 cycle?



DO follow!

Model Components

C_M	//	*model classes*
Q_M	//	*model element instances*
$F_M : C_M \leftrightarrow \text{ident}$	//	*fields(no shadowing)*
$R_n \subseteq F_M$	//	*Selection of fields, defining traversal*
V_n^{G}	//	*Base visitor for R_n, realizing traversal*
$V_n^{\mathrm{U1}}, V_n^{\mathrm{U2}}, \ldots$	//	*User def'd visitors, realizing semantics*
V_n	//	*All visitors following selection n*
V_M	//	*All visitors*

Visitor Default Behaviour

$$\begin{array}{lcl} \mathcal{S} & // & \textit{some state space} \\ \textit{transformationType} & = & V_M \times Q_M \rightarrow (\mathcal{S} \nrightarrow \mathcal{S}) \end{array}$$

Visitor Default Behaviour

$$
\begin{array}{lcl}
\mathcal{S} & // & \textit{some state space} \\
\textit{transformationType} & = & V_M \times Q_M \to (\mathcal{S} \nrightarrow \mathcal{S}) \\
 & & \\
\texttt{match} & : & \textit{transformationType}
\end{array}
$$

Visitor Default Behaviour

$$
\begin{array}{lcl}
\mathcal{S} & // & \textit{some state space} \\
\textit{transformationType} & = & V_M \times Q_M \to (\mathcal{S} \nrightarrow \mathcal{S}) \\
 & & \\
\texttt{match} & : & \textit{transformationType} \\
\texttt{match}\ (v, q) & = & (\textit{getAction}(v, \texttt{getClass}\ q))(v, q) \\
\textit{getAction} & : & (V_M \times C_M) \to \textit{transformationType}
\end{array}
$$

Visitor Default Behaviour

$$
\begin{array}{lcl}
\mathcal{S} & // & \textit{some state space} \\
\textit{transformationType} & = & V_M \times Q_M \to (\mathcal{S} \nrightarrow \mathcal{S}) \\
 & & \\
\texttt{match}, \texttt{descend} & : & \textit{transformationType} \\
\texttt{match}\ (v, q) & = & (\textit{getAction}(v, \texttt{getClass}\ q))(v, q) \\
\textit{getAction} & : & (V_M \times C_M) \to \textit{transformationType} \\
\textit{getAction}(V_n^{\mathrm{G}}, c) & = & \texttt{descend}_{c,n} \\
R_n(\!|\{c\}|\!) & = & \{f_1, \ldots, f_k\} \\
\texttt{descend}_{c,n}(v, q) & = & \texttt{descend}_{\texttt{super}(c),n}(v, q) \\
\quad \fatsemi \textit{matchField}(v, q, f_1) \fatsemi \ldots \fatsemi \textit{matchField}(v, q, f_k) & &
\end{array}
$$

Visitor Default Behaviour

$$
\begin{array}{lcl}
\mathcal{S} & // & \textit{some state space} \\
\textit{transformationType} & = & V_M \times Q_M \to (\mathcal{S} \nrightarrow \mathcal{S}) \\
 & & \\
\texttt{match}, \texttt{descend} & : & \textit{transformationType} \\
\texttt{match}\ (v, q) & = & (\textit{getAction}(v, \texttt{getClass}\ q))(v, q) \\
\textit{getAction} & : & (V_M \times C_M) \to \textit{transformationType} \\
\textit{getAction}(V_n^{\mathrm{G}}, c) & = & \texttt{descend}_{c,n} \\
R_n(\!|\{c\}|\!) & = & \{f_1, \ldots, f_k\} \\
\texttt{descend}_{c,n}(v, q) & = & \texttt{descend}_{\texttt{super}(c),n}(v, q) \\
\quad \fatsemi \textit{matchField}(v, q, f_1) \fatsemi \ldots \fatsemi \textit{matchField}(v, q, f_k) & & \\
\textit{referred}(\textit{get}(q, f_x)) & = & \{q_1, \ldots, q_m\} \\
\textit{matchField}(v, q, f_x) & = & \texttt{match}(v, q_1) \fatsemi \ldots \fatsemi \texttt{match}(v, q_m)
\end{array}
$$

Visitor Default Behaviour – Properties

Properties

of the default visitors V_n^{G} :

- No cycles $\longrightarrow$ Termination
- Cycles $\longrightarrow$ No termination
- state ($\in \mathcal{S}$) is not altered

User Defined Visitors

$$
\begin{array}{lcl}
V_n^{\mathrm{U}m} & : & V_n \times (C_M \nrightarrow \mathit{transformationType}) \\
\pi_1 = \texttt{baseVisitor} & : & V_n^{\mathrm{U}m} \to V_n \\
\pi_2 = \mathit{actionDefs} & : & V_n^{\mathrm{U}m} \to (C_M \nrightarrow \mathit{transformationType}) \\
\mathit{getAction}(V_n^{\mathrm{U}m}, \_) & = & \mathit{getAction}(\texttt{baseVisitor}(V_n^{\mathrm{U}m}), \_) \\
 & & \quad \oplus \mathit{actionDefs}(V_n^{\mathrm{U}m}) \\
\mathit{defedClasses} & : & V_M \to \mathbb{P}\, C_M \\
\mathit{defedClasses}(V_n^{\mathrm{G}}) & = & \{\} \\
\mathit{defedClasses}(v : V_n^{\mathrm{U}m}) & = & \\
 & & \quad \mathrm{dom}\, \mathit{actionDefs}(v) \cup \mathit{defedClasses}(\texttt{baseVisitor}(v))
\end{array}
$$

User Defined Visitors

$$
\begin{array}{lcl}
V_n^{\mathrm{U}m} & : & V_n \times (C_M \nrightarrow \mathit{transformationType}) \\
\pi_1 = \texttt{baseVisitor} & : & V_n^{\mathrm{U}m} \to V_n \\
\pi_2 = \mathit{actionDefs} & : & V_n^{\mathrm{U}m} \to (C_M \nrightarrow \mathit{transformationType}) \\
\mathit{getAction}(V_n^{\mathrm{U}m}, \_) & = & \mathit{getAction}(\texttt{baseVisitor}(V_n^{\mathrm{U}m}), \_) \\
 & & \quad \oplus \mathit{actionDefs}(V_n^{\mathrm{U}m}) \\
\mathit{defedClasses} & : & V_M \to \mathbb{P}\, C_M \\
\mathit{defedClasses}(V_n^{\mathrm{G}}) & = & \{\} \\
\mathit{defedClasses}(v : V_n^{\mathrm{U}m}) & = & \\
 & & \quad \mathrm{dom}\, \mathit{actionDefs}(v) \cup \mathit{defedClasses}(\texttt{baseVisitor}(v))
\end{array}
$$

User Defined Visitors

$$
\begin{array}{lcl}
V_n^{\mathrm{U}m} & : & V_n \times (C_M \nrightarrow \mathit{transformationType}) \\
\pi_1 = \texttt{baseVisitor} & : & V_n^{\mathrm{U}m} \to V_n \\
\pi_2 = \mathit{actionDefs} & : & V_n^{\mathrm{U}m} \to (C_M \nrightarrow \mathit{transformationType}) \\
\mathit{getAction}(V_n^{\mathrm{U}m}, \_) & = & \mathit{getAction}(\texttt{baseVisitor}(V_n^{\mathrm{U}m}), \_) \\
 & & \quad \oplus \mathit{actionDefs}(V_n^{\mathrm{U}m}) \\
\mathit{defedClasses} & : & V_M \to \mathbb{P}\, C_M \\
\mathit{defedClasses}(V_n^{\mathrm{G}}) & = & \{\} \\
\mathit{defedClasses}(v : V_n^{\mathrm{U}m}) & = & \\
 & & \mathrm{dom}\, \mathit{actionDefs}(v) \cup \mathit{defedClasses}(\texttt{baseVisitor}(v))
\end{array}
$$

Optimization

$$includeFieldsOf, decClassToDecClass_n, connected_n : C_M \leftrightarrow C_M$$

$$\begin{array}{lcl} includeFieldsOf & = & \mathtt{super}^* \cup (\mathtt{super}^*)^\sim \\ decClassToDecClass_n & = & R_n \fatsemi fieldTypes \fatsemi containedClasses \\ containedClasses & : & \mathcal{T} \leftrightarrow C_M \\ \mathcal{T} & ::= & C_M \mid \mathcal{C}_0 \mid \mathcal{T}_{\text{prim}} \\ & & \mid \mathtt{OPT}\,\mathcal{T} \mid \mathtt{SEQ}\,\mathcal{T} \mid \mathtt{SET}\,\mathcal{T} \\ & & \mid \mathtt{MAP}\,\mathcal{T}\ \mathtt{TO}\,\mathcal{T} \mid \mathtt{REL}\,\mathcal{T}\ \mathtt{TO}\,\mathcal{T} \end{array}$$

$$connected_n = includeFieldsOf \fatsemi decClassToDecClass_n \fatsemi includeFieldsOf$$

Optimization

$$includeFieldsOf, decClassToDecClass_n, connected_n : C_M \leftrightarrow C_M$$

$$includeFieldsOf = \texttt{super}^* \cup (\texttt{super}^*)^\sim$$

$$decClassToDecClass_n = R_n \fatsemi fieldTypes \fatsemi containedClasses$$

$$containedClasses : \mathcal{T} \leftrightarrow C_M$$

$$\mathcal{T} ::= C_M \mid \mathcal{C}_0 \mid \mathcal{T}_{\text{prim}} \mid \texttt{OPT}\,\mathcal{T} \mid \texttt{SEQ}\,\mathcal{T} \mid \texttt{SET}\,\mathcal{T} \mid \texttt{MAP}\,\mathcal{T}\,\texttt{TO}\,\mathcal{T} \mid \texttt{REL}\,\mathcal{T}\,\texttt{TO}\,\mathcal{T}$$

$$connected_n = includeFieldsOf \fatsemi decClassToDecClass_n \fatsemi includeFieldsOf$$

Optimization

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{includeFieldsOf}, \mathit{decClassToDecClass}_n, \mathit{connected}_n : C_M \leftrightarrow C_M} \\
\mathit{includeFieldsOf} & = & \mathtt{super}^* \cup (\mathtt{super}^*)^\sim \\
\mathit{decClassToDecClass}_n & = & R_n \fatsemi \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \\
\mathit{containedClasses} & : & \mathcal{T} \leftrightarrow C_M \\
\mathcal{T} & ::= & C_M \mid \mathcal{C}_0 \mid \mathcal{T}_{\mathrm{prim}} \\
 & & \mid \mathtt{OPT}\,\mathcal{T} \mid \mathtt{SEQ}\,\mathcal{T} \mid \mathtt{SET}\,\mathcal{T} \\
 & & \mid \mathtt{MAP}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \mid \mathtt{REL}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \\
\mathit{connected}_n & = & \\
\multicolumn{3}{r}{\mathit{includeFieldsOf} \fatsemi \mathit{decClassToDecClass}_n \fatsemi \mathit{includeFieldsOf}}
\end{array}
$$

Optimization

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{includeFieldsOf}, \mathit{decClassToDecClass}_n, \mathit{connected}_n : C_M \leftrightarrow C_M} \\
\mathit{includeFieldsOf} & = & \mathtt{super}^* \cup (\mathtt{super}^*)^\sim \\
\mathit{decClassToDecClass}_n & = & R_n \fatsemi \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \\
\mathit{containedClasses} & : & \mathcal{T} \leftrightarrow C_M \\
\mathcal{T} & ::= & C_M \mid \mathcal{C}_0 \mid \mathcal{T}_{\text{prim}} \\
 & & \mid \mathtt{OPT}\,\mathcal{T} \mid \mathtt{SEQ}\,\mathcal{T} \mid \mathtt{SET}\,\mathcal{T} \\
 & & \mid \mathtt{MAP}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \mid \mathtt{REL}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \\
\mathit{connected}_n & = & \\
\multicolumn{3}{r}{\mathit{includeFieldsOf} \fatsemi \mathit{decClassToDecClass}_n \fatsemi \mathit{includeFieldsOf}}
\end{array}
$$

Optimization

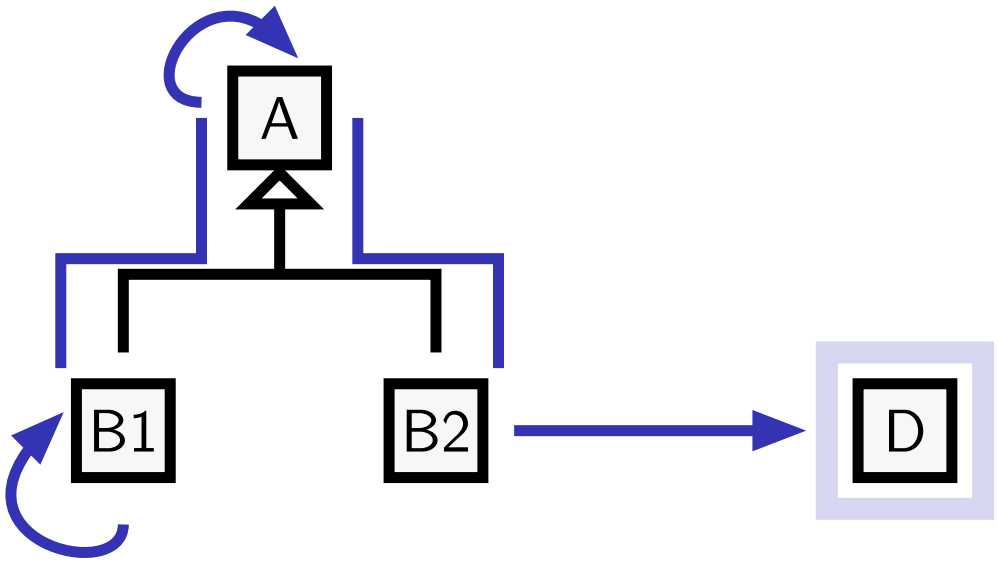
$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{includeFieldsOf}, \mathit{decClassToDecClass}_n, \mathit{connected}_n : C_M \leftrightarrow C_M} \\
\mathit{includeFieldsOf} & = & \mathtt{super}^* \cup (\mathtt{super}^*)^\sim \\
\mathit{decClassToDecClass}_n & = & R_n \fatsemi \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \\
\mathit{containedClasses} & : & \mathcal{T} \leftrightarrow C_M \\
\mathcal{T} & ::= & C_M \mid \mathcal{C}_0 \mid \mathcal{T}_{\mathrm{prim}} \\
 & & \mid \mathtt{OPT}\,\mathcal{T} \mid \mathtt{SEQ}\,\mathcal{T} \mid \mathtt{SET}\,\mathcal{T} \\
 & & \mid \mathtt{MAP}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \mid \mathtt{REL}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \\
\mathit{connected}_n & = & \\
\multicolumn{3}{r}{\mathit{includeFieldsOf} \fatsemi \mathit{decClassToDecClass}_n \fatsemi \mathit{includeFieldsOf}}
\end{array}
$$

Optimization

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{includeFieldsOf}, \mathit{decClassToDecClass}_n, \mathit{connected}_n : C_M \leftrightarrow C_M} \\
\mathit{includeFieldsOf} & = & \mathtt{super}^* \cup (\mathtt{super}^*)^\sim \\
\mathit{decClassToDecClass}_n & = & R_n \fatsemi \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \\
\mathit{containedClasses} & : & \mathcal{T} \leftrightarrow C_M \\
\mathcal{T} & ::= & C_M \mid \mathcal{C}_0 \mid \mathcal{T}_{\mathrm{prim}} \\
 & & \mid \mathtt{OPT}\,\mathcal{T} \mid \mathtt{SEQ}\,\mathcal{T} \mid \mathtt{SET}\,\mathcal{T} \\
 & & \mid \mathtt{MAP}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \mid \mathtt{REL}\,\mathcal{T}\,\mathtt{TO}\,\mathcal{T} \\
\mathit{connected}_n & = & \\
\multicolumn{3}{r}{\mathit{includeFieldsOf} \fatsemi \mathit{decClassToDecClass}_n \fatsemi \mathit{includeFieldsOf}}
\end{array}
$$

Is it necessary to follow a certain association?

connected is mixed from associations and super class relation



DO follow!

Optimization

$$\texttt{fieldFlags} \;:\; F_M \leftrightarrow V_n^{\mathrm{U}m}$$
$$= \; \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \fatsemi \mathit{connected}_n{}^* \fatsemi (\mathit{defedClasses}\ V_n^{\mathrm{U}m})^{\sim}$$

$$\mathit{matchField}(v, q, f_x)$$
$$= \begin{cases} \texttt{match}(v, q_1) \fatsemi \ldots \fatsemi \texttt{match}(v, q_m) & \text{if } \texttt{fieldFlags}\,(f_x, v) \\ \mathrm{id} & \text{otherwise} \end{cases}$$

Optimization

$$\texttt{fieldFlags} \;:\; F_M \leftrightarrow V_n^{\mathrm{U}m}$$
$$= \; \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \fatsemi \mathit{connected}_n{}^* \fatsemi (\mathit{defedClasses}\ V_n^{\mathrm{U}m})^{\sim}$$

$$\mathit{matchField}(v, q, f_x)$$
$$= \begin{cases} \texttt{match}(v, q_1) \fatsemi \ldots \fatsemi \texttt{match}(v, q_m) & \text{if } \texttt{fieldFlags}\ (f_x, v) \\ \mathrm{id} & \text{otherwise} \end{cases}$$

Optimization

$$\texttt{fieldFlags} \;:\; F_M \leftrightarrow V_n^{\mathrm{U}m}$$
$$= \; \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \fatsemi \mathit{connected}_n{}^* \fatsemi (\mathit{defedClasses}\ V_n^{\mathrm{U}m})^\sim$$

$$\mathit{matchField}(v, q, f_x)$$
$$= \begin{cases} \texttt{match}(v, q_1) \fatsemi \ldots \fatsemi \texttt{match}(v, q_m) & \text{if } \texttt{fieldFlags}\ (f_x, v) \\ \mathrm{id} & \text{otherwise} \end{cases}$$

code generation time ("compile time")

Optimization

$$\texttt{fieldFlags} \;:\; F_M \leftrightarrow V_n^{\mathrm{U}m}$$
$$= \; \mathit{fieldTypes} \fatsemi \mathit{containedClasses} \fatsemi \mathit{connected}_n{}^* \fatsemi (\mathit{defedClasses}\ V_n^{\mathrm{U}m})^\sim$$

$$\mathit{matchField}(v, q, f_x)$$
$$= \begin{cases} \texttt{match}(v, q_1) \fatsemi \ldots \fatsemi \texttt{match}(v, q_m) & \text{if } \texttt{fieldFlags}\ (f_x, v) \\ \mathrm{id} & \text{otherwise} \end{cases}$$

code generation time ("compile time") vs. class loading time

Typical components of meta–tools

meta–tools, a collection of tools for meta programming and compiler construction. Contains (among others !-) :

- `format` a framework for human-readable text and code layout, enhancing HUGHES's pretty-printing combinators substantially
- `metajava` partial model of Java, for source code generation, seamlessly integrated with reflection
- `tdom` a *strictly typed* XML document object model
- `option` compiler for command-line style and GUI style parametrization of applications
- `umod` compiler for Java models from a mathematical description
  (read "μ-mod" i.e. "micro model")

Typical components of meta–tools

meta–tools, a collection of tools for meta programming and compiler construction. Contains (among others !-) :

format a framework for human-readable text and code layout, enhancing HUGHES's pretty-printing combinators substantially

metajava partial model of Java, for source code generation, seamlessly integrated with reflection

tdom a *strictly typed* XML document object model

option compiler for command-line style and GUI style parametrization of applications

umod compiler for Java models from a mathematical description
(read "μ-mod" i.e. "micro model")

Typical components of meta–tools

meta–tools, a collection of tools for meta programming and compiler construction. Contains (among others !-) :

- **format** a framework for human-readable text and code layout, enhancing HUGHES's pretty-printing combinators substantially
- **metajava** partial model of Java, for source code generation, seamlessly integrated with reflection
- **tdom** a *strictly typed* XML document object model
- **option** compiler for command-line style and GUI style parametrization of applications
- **umod** compiler for Java models from a mathematical description (read "μ-mod" i.e. "micro model")

Typical components of meta–tools

meta–tools, a collection of tools for meta programming and compiler construction. Contains (among others !-) :

- **format** a framework for human-readable text and code layout, enhancing HUGHES's pretty-printing combinators substantially
- **metajava** partial model of Java, for source code generation, seamlessly integrated with reflection
- **tdom** a *strictly typed* XML document object model
- **option** compiler for command-line style and GUI style parametrization of applications
- **umod** compiler for Java models from a mathematical description (read "μ-mod" i.e. "micro model")

Typical components of meta–tools

meta–tools, a collection of tools for meta programming and compiler construction. Contains (among others !-) :

- **format** a framework for human-readable text and code layout, enhancing HUGHES's pretty-printing combinators substantially
- **metajava** partial model of Java, for source code generation, seamlessly integrated with reflection
- **tdom** a *strictly typed* XML document object model
- **option** compiler for command-line style and GUI style parametrization of applications
- **umod** compiler for Java models from a mathematical description
  (read "μ-mod" i.e. "micro model")

Typical components of meta–tools

meta–tools, a collection of tools for meta programming and compiler construction. Contains (among others !-) :

- **format** a framework for human-readable text and code layout, enhancing HUGHES's pretty-printing combinators substantially
- **metajava** partial model of Java, for source code generation, seamlessly integrated with reflection
- **tdom** a *strictly typed* XML document object model
- **option** compiler for command-line style and GUI style parametrization of applications
- **umod** compiler for Java models from a mathematical description
  (read "μ-mod" i.e. "micro model")

umod features

umod : generate Java code for general-purpose data models

- the classes for the model's elements
- safe constructors and setter methods (primarily w.r.t. the illegal value null)
- various methods for visualization and (de-)serialization and
- different kinds of visitors
- Still missing:Pattern matching

umod features

umod : generate Java code for general-purpose data models

- the classes for the model's elements
- safe constructors and setter methods (primarily w.r.t. the illegal value null)
- various methods for visualization and (de-)serialization and
- different kinds of visitors
- Still missing:Pattern matching

umod features

umod : generate Java code for general-purpose data models

- the classes for the model's elements
- safe constructors and setter methods (primarily w.r.t. the illegal value `null`)
- various methods for visualization and (de-)serialization and
- different kinds of visitors
- Still missing:Pattern matching

umod features

umod : generate Java code for general-purpose data models

- the classes for the model's elements
- safe constructors and setter methods (primarily w.r.t. the illegal value `null`)
- various methods for visualization and (de-)serialization and
- different kinds of visitors
- Still missing:Pattern matching

umod features

umod : generate Java code for general-purpose data models

- the classes for the model's elements
- safe constructors and setter methods (primarily w.r.t. the illegal value `null`)
- various methods for visualization and (de-)serialization and
- different kinds of visitors
- Still missing:Pattern matching

umod features

umod : generate Java code for general-purpose data models

- the classes for the model's elements
- safe constructors and setter methods (primarily w.r.t. the illegal value `null`)
- various methods for visualization and (de-)serialization and
- different kinds of visitors
- Still missing:Pattern matching

umod example

```
MODEL M =


  TOPLEVEL CLASS
    A

    | B1

    | B2

    D

END MODEL
```

umod example

```
MODEL M =


  TOPLEVEL CLASS
    A ABSTRACT

    | B1 ALGEBRAIC

    | B2

    D

END MODEL
```

umod example

```
MODEL M =

  TOPLEVEL CLASS
    A ABSTRACT
          a1 A
          a2 int <-> B1
    | B1 ALGEBRAIC

          b1_2 SEQ (B1*int)
    | B2
          b2   (string+int)->OPT(A <->OPT D)
    D
          d int
END MODEL
```

umod example

```
MODEL M =
  EXT point = java.awt.geom.Point2D



  TOPLEVEL CLASS
    A ABSTRACT
        a1 A
        a2 int <-> B1
    | B1 ALGEBRAIC
        b1_1 OPT point
        b1_2 SEQ (B1*int)
    | B2
        b2   (string+int)->OPT(A <->OPT D)
    D EXTENDS point
        d int
END MODEL
```

umod example

```
MODEL M =
  EXT point = java.awt.geom.Point2D



  TOPLEVEL CLASS
    A ABSTRACT
        a1 A                ! C 0/0          ;
        a2 int <-> B1
    | B1 ALGEBRAIC
        b1_1 OPT point
        b1_2 SEQ (B1*int)
    | B2
        b2   (string+int)->OPT(A <->OPT D)
    D EXTENDS point
        d int = "17"
END MODEL
```

umod example

```
MODEL M =
  EXT point = java.awt.geom.Point2D



  TOPLEVEL CLASS
    A ABSTRACT
        a1 A                ! C 0/0  V     1/1 ;
        a2 int <-> B1       !        V 0/0 1/0 ;
    | B1 ALGEBRAIC
        b1_1 OPT point      !        V     1/0 ;
        b1_2 SEQ (B1*int)   !        V 0/0     ;
    | B2
        b2   (string+int)->OPT(A <->OPT D)! V 0/0 1/0 RR;
    D EXTENDS point
        d int = "17"
END MODEL
```

umod example

```
MODEL M =
  EXT point = java.awt.geom.Point2D
  VISITOR 0 Simple ;
  VISITOR 1 Rewrite IS REWRITER ;
  VISITOR 1 Visitor MULTIPHASE ;

  TOPLEVEL CLASS
    A ABSTRACT
        a1 A               ! C 0/0  V     1/1 ;
        a2 int <-> B1      !        V 0/0 1/0 ;
    | B1 ALGEBRAIC
        b1_1 OPT point     !        V     1/0 ;
        b1_2 SEQ (B1*int)  !        V 0/0     ;
    | B2
        b2   (string+int)->OPT(A <->OPT D)! V 0/0 1/0 RR;
    D EXTENDS point
        d int = "17"
END MODEL
```

- ToFu = "Total Functional Language"
- Executable framework for the *Trace Function Method*
  - Formal black-box component behaviour description language
  - Tabular expression format (aka "Parnas Tables")
  - Combines partial functions and total logic (cf. IEEE 754)
  - Trancón and Parnas (IFL2007/extended papers 2008)
- Compiler core:
  - (Nearly) complete analysis and translation of ToFu to Java, including sophisticated optimization:
    10 634 lines of handcoded Java.
  - 560 lines of `umod/xantlr`, generating additional 18 598 lines of Java

- ToFu = "Total Functional Language"
- Executable framework for the *Trace Function Method*
  - Formal black-box component behaviour description language
  - Tabular expression format (aka "Parnas Tables")
  - Combines partial functions and total logic (cf. IEEE 754)
  - Trancón and Parnas (IFL2007/extended papers 2008)
- Compiler core:
  - (Nearly) complete analysis and translation of ToFu to Java, including sophisticated optimization:
    10 634 lines of handcoded Java.
  - 560 lines of `umod/xantlr`, generating additional 18 598 lines of Java

- ToFu = "Total Functional Language"
- Executable framework for the *Trace Function Method*
  - Formal black-box component behaviour description language
  - Tabular expression format (aka "Parnas Tables")
  - Combines partial functions and total logic (cf. IEEE 754)
  - Trancón and Parnas (IFL2007/extended papers 2008)
- Compiler core:
  - (Nearly) complete analysis and translation of ToFu to Java, including sophisticated optimization:
    10 634 lines of handcoded Java.
  - 560 lines of `umod/xantlr`, generating additional 18 598 lines of Java

ToFu example

```
MODEL ToFu =
TOPLEVEL CLASS
  Container ABSTRACT
       items SimpleName -> Item       ! V 0/0 ;
       imports SET Import
  | Section
  Item ABSTRACT
       declaration Declaration        ! V 0/0 ;
       visibility Visibility
       fullname QualifiedName
//| ...
  | Subsection
       section Section                ! V 0/1 ;
  | Import
  Specification ABSTRACT
  | Declaration
       // ...
  | Definition
       // ...
END MODEL
```

ToFu example

```
public Set<QName> collectImports(Container context,
                                 final Visibility minVisib) {









}
```

ToFu example

```
public Set<QName> collectImports(Container context,
                                 final Visibility minVisib) {
  final Set<QName> result = new HashSet<QName>() ;
  new Visitor() {




    }
  }.match(context) ;
  return result ;
}
```

ToFu example

```
public Set<QName> collectImports(Container context,
                                 final Visibility minVisib) {
  final Set<QName> result = new HashSet<QName>() ;
  new Visitor() {
    @Override protected void action(Import imp) {
      if (imp.get_visibility().compareTo(minVisib) >= 0)
        result.add(imp.get_fullname()) ;
    }



    }
  }.match(context) ;
  return result ;
}
```

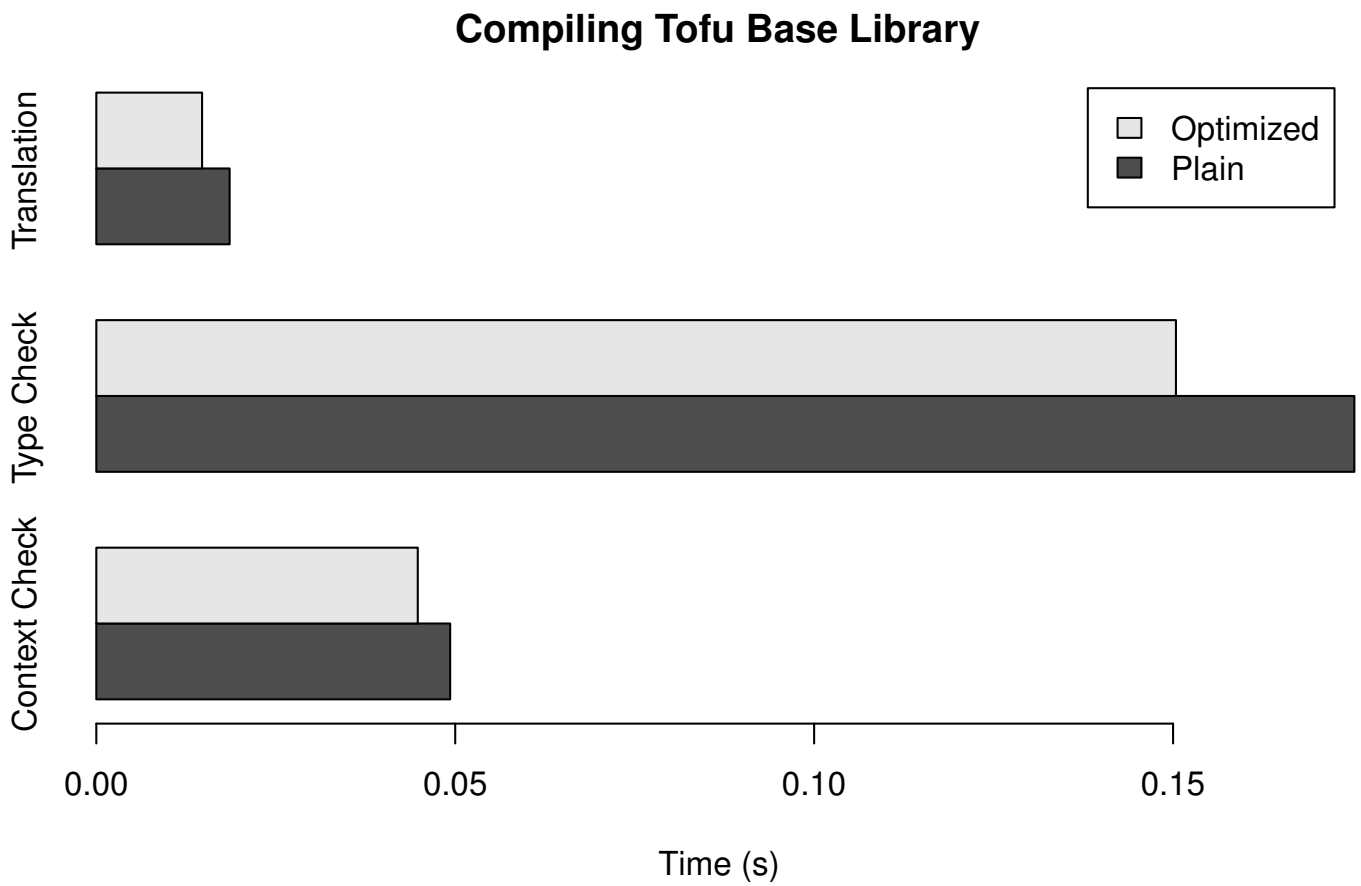
ToFu example

```
public Set<QName> collectImports(Container context,
                                 final Visibility minVisib) {
  final Set<QName> result = new HashSet<QName>() ;
  new Visitor() {
    @Override protected void action(Import imp) {
      if (imp.get_visibility().compareTo(minVisib) >= 0)
        result.add(imp.get_fullname()) ;
    }
    @Override protected void action(Declaration decl) {}
    @Override protected void action(Definition def) {}
    @Override protected void action(Subsection sub) {
      // select only top-level imports
    }
  }.match(context) ;
  return result ;
}
```

ToFu example

```
public Set<QName> collectImports(Container context,
                                 final Visibility minVisib) {
  final Set<QName> result = new HashSet<QName>() ;
  new Visitor() {
    @Override protected void action(Import imp) {
      if (imp.get_visibility().compareTo(minVisib) >= 0)
        result.add(imp.get_fullname()) ;
    }


    @Override protected void action(Subsection sub) {
      // select only top-level imports
    }
  }.match(context) ;
  return result ;
}
```


Compiling Tofu Base Library
Optimized
Plain
Translation
Type Check
Context Check
0.00
0.05
0.10
0.15
Time (s)

More . . .

- . . . in the proceedings
- . . . meta–tools users' guide at `http://bandm.eu`

More . . .

- . . . in the proceedings
- . . . $^{\text{meta}}$–tools users' guide at `http://bandm.eu`