Static Analysis of AXML Services

Serge Abiteboul Luc Segoufin Victor Vianu

Bordeaux, October 2007

Organization

- Introduction: motivation and goals
- The GAXML model (AXML with guards)
- Temporal properties
- Results

Motivation

- Documents evolving in time via function calls
  - That compute locally
  - That receive data from external sources and thereby interact with their environment
- This is in the spirit of business artifacts that are used to model activities
  - The states of documents or of portions of documents correspond to process states in workflow systems
- We want to reason about such documents
  - Verify temporal properties

Goals

- Model such documents using GAXML
  - Some nonmonotonicity (vs. positive AXML)
  - But very limited so that decidable
- Model properties of the evolution using temporal logics

- Results: boundary of tractability of verification of temporal properties

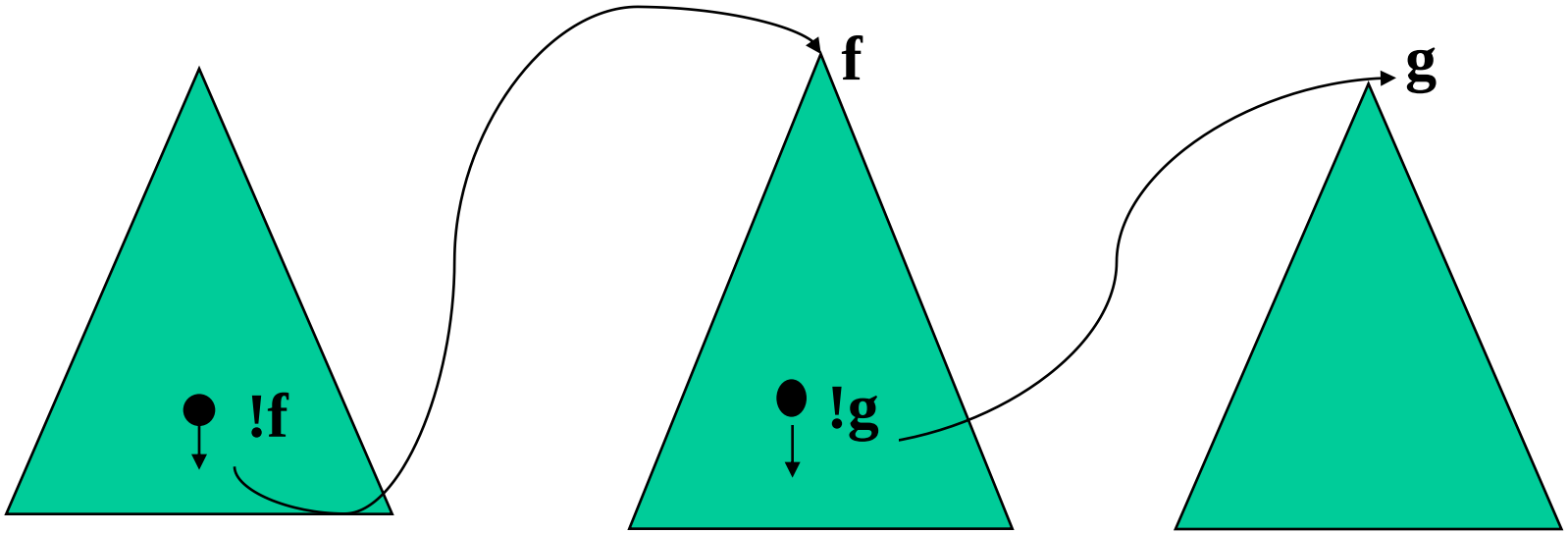
Basics of the model

- Unordered labeled trees and set semantics
  - No isomorphic sibling subtrees
  - Internal nodes are labeled by tags
  - Leaves are labeled by tags, data, or function symbols
- Trees with constraints of 2 kinds
  - DTDs adapted to unordered trees
  - Boolean combinations of tree patterns

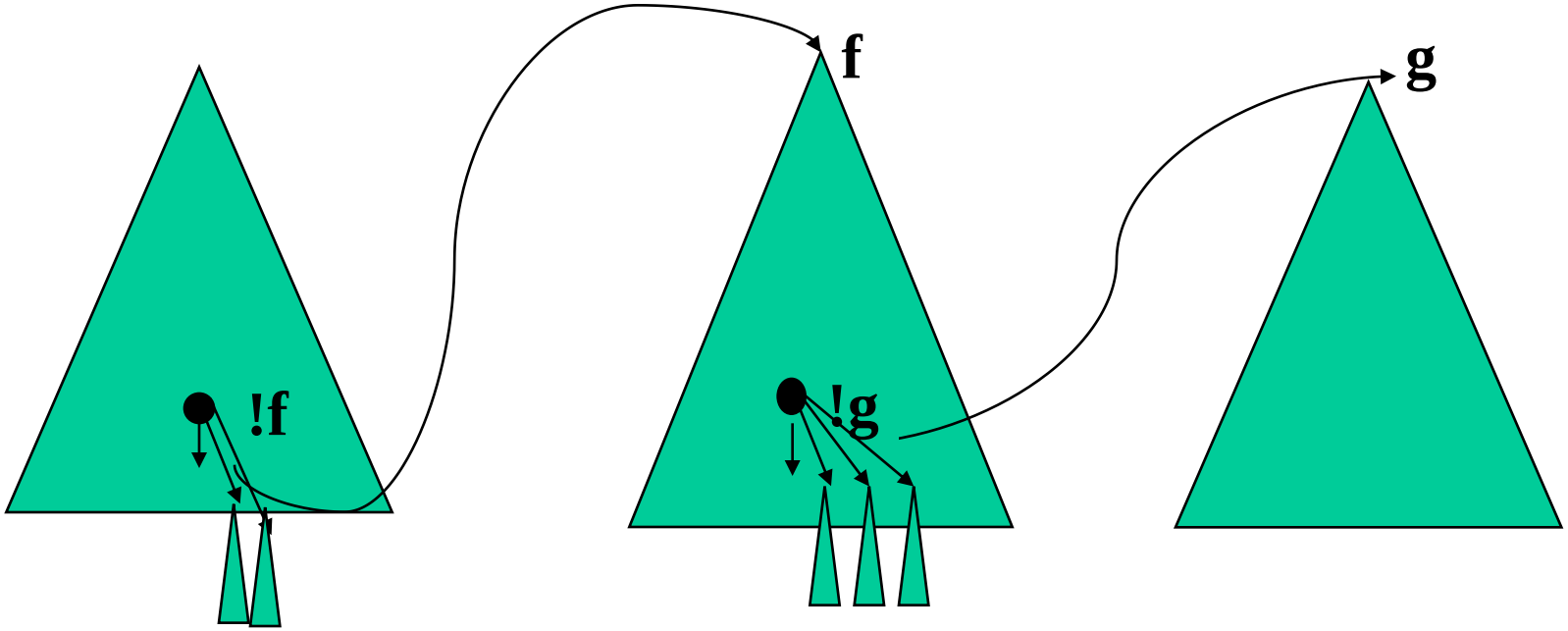
  I know this is not very elegant but accept it for now

GAXML vs. positive AXML

- Distinction between call and return of functions
  1. A call is fired (data is passed as argument)
  2. It is evaluated in some workspace
  3. The evaluation terminates and the result is returned
- Calls are controlled by guards
  - Call guards and return guards
  - Boolean combinations of tree patterns = *BCTP*
- Limited monmonotone features
  - ***Nonpositive guards***
  - Some functions terminate (can be captured by guards)



- Function calls
  - ?f : intentional call
  - !f : call has been activated
  - f : workspace where the call is evaluated



- Data is returned
- Function call terminates – call disappears
- Continuous call – call remains
- Run: infinite sequence of consecutive instances satisfying the static constraints

GAXML schema specification

- Some document names
  - Possibly with static constraints: DTD & BCTP
- Internal functions specification
- External functions specifications

Note: A single peer in the current model. This can be easily introduced if desired.

Internal Function Specification

- Kind: continuous or non-continuous
- Call and return guards: BCTP

Two tree pattern queries

- Argument query: defines data to pass as argument
- Return query: defines data to return as result

External ~~Internal~~ Function Specification

- Kind: continuous or non-continuous
- Call ~~and return~~ guards: BCTP

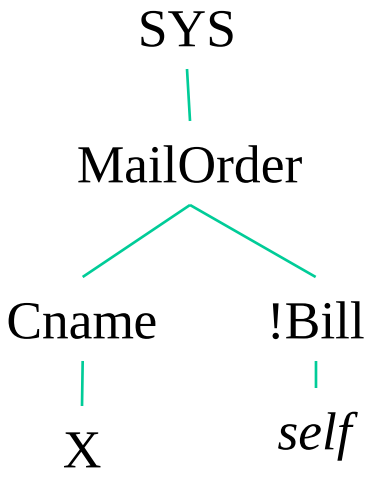
Two tree pattern queries

- Argument query: defines data to pass as argument
- ~~Return query: defines data to return as result~~

Intuition: unknown specification or interaction with external users/applications

What is a (relative) tree pattern?
(guards are Boolean combination of TPs)

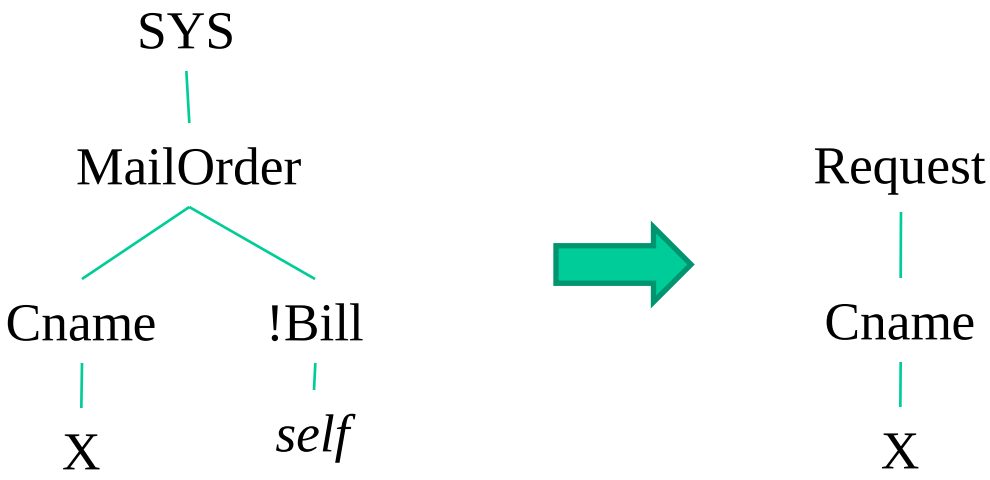
- Tree
  - Internal nodes are labeled by tags
  - Leaves are labeled by tags, function symbols, data values or **variables**
- Edges labeled by / or // (descendant)
- Condition on data variables
  - Boolean combination of (in)equalities X = Y, X = a
- Relative: also uses "*self*"
  - *self* is a specific tag
  - Refers to a specific node



What is a query?
(for argument and result queries)

Expression **Body → Head** where:

- **Body** is a tree pattern
- **Head** is labeled tree (possibly with variables from Body)
- Informal semantics (classical) = forest



Example: Mail Order

The database consists of Catalog providing prices

A customer initializes a MailOrder

User name (Cname), product name (Pname), system supplied OrderId

The MailOrder systems sends out an Invoice with

Cname, Pname, price of product from Catalog

The customer responds with a Payment with

Pname, amount, and kind of payment (credit or check)

If payment is incorrect (amount not equal to catalog price)

the product is rebilled - !Bill is a continous function

If the payment is correct and by check

the product is delivered and we terminate

If the payment is correct and by card

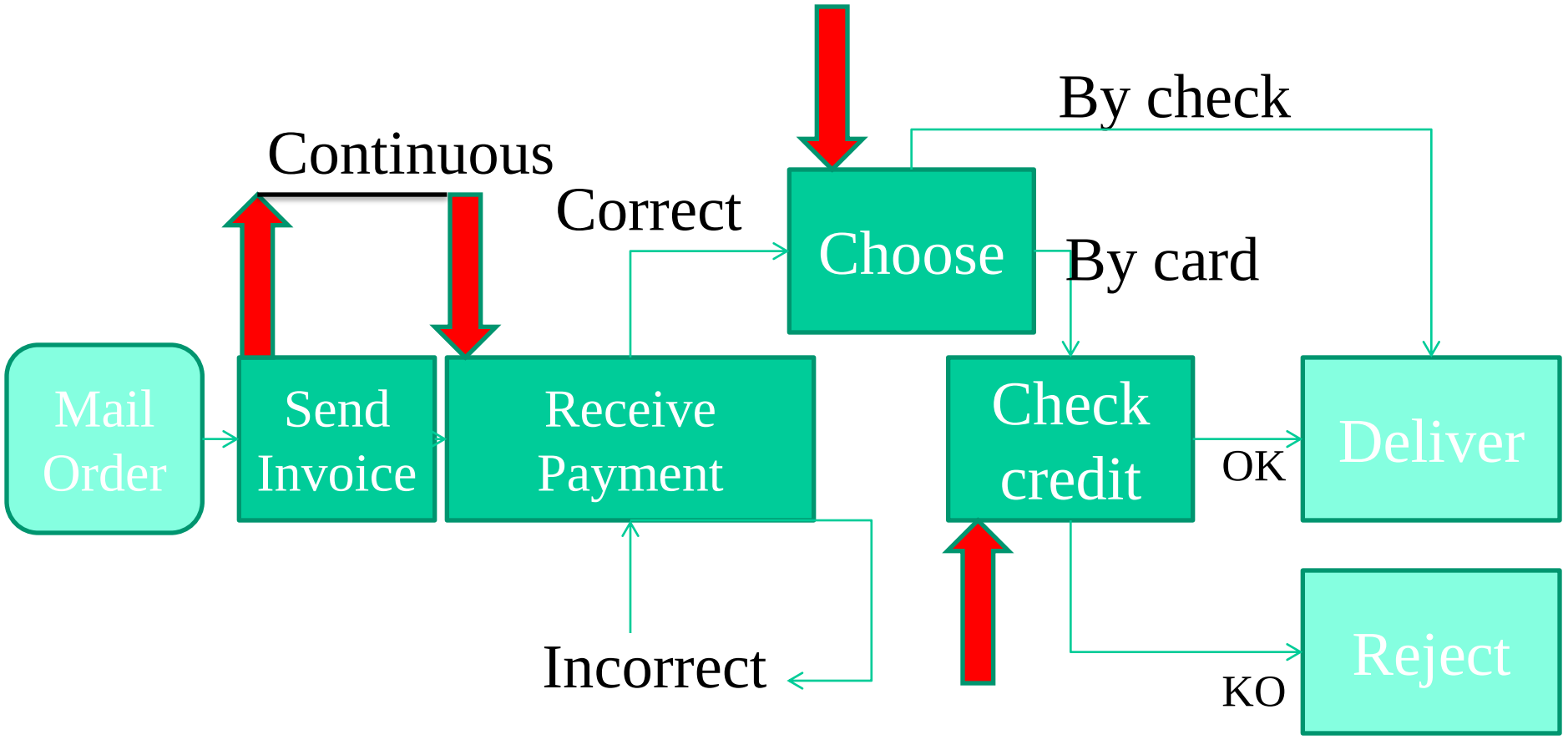
the system asks for a CreditCheck

if the credit rating is Good,

the product is delivered and we terminate

otherwise the order is rejected and we terminate

Workflow for MailOrder



Mail Order Schema

The DTD

SYS: Catalog (!MailOrder XOR ?MailOrder) (MailOrder)*

Catalog: (Product) *

Product: Pname Price

MailOrder: Order-id Cname Pname
AND (!Bill XOR ?Bill) (Payment) *
AND (!Credit-Check XOR ?CreditCheck XOR CreditRating)
AND (!Deliver XOR ?Deliver XOR Delivered)
AND (!Reject XOR ?Reject XOR Rejected)

Payment: Pname Amount Kind

Kind: Credit XOR Check

CreditRating: Good XOR Bad

Order-id, Price, Pname, Cname, Pname, Amount: **dom**

Rejected, Delivered, Good, Bad: empty (leaves)

Shorthand: A stands for |A| = 1, A* stands for |A| >= 0, concat is AND

Guards

Call guard: block a function from the client/caller perspective

- Do not call !deliver unless

  Paid-check-ok OR Credit-ok

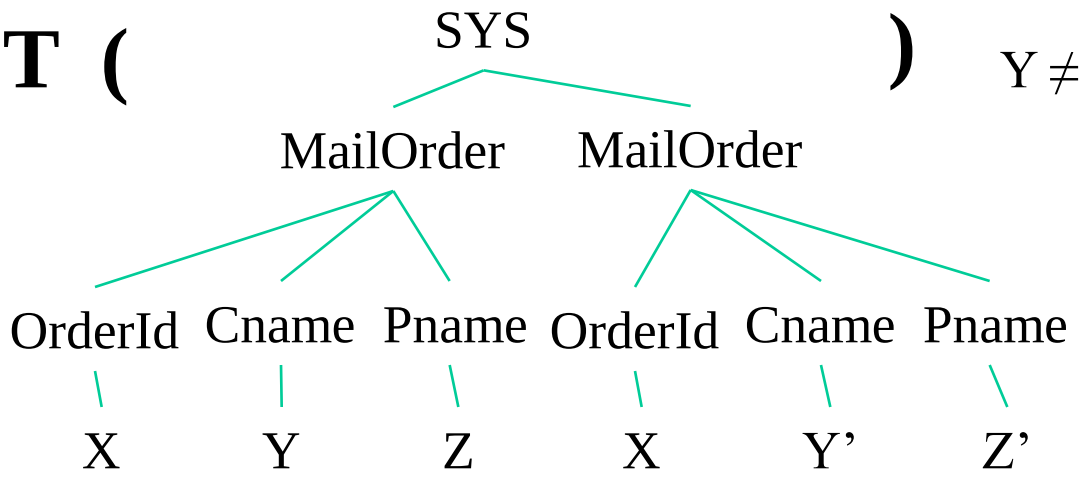
- Stop accepting return values from !Bill

  After the first correct payment

Return guard: block from the server perspective (called one)

Schema (cont'd): Data constraints

- There are no distinct MailOrders with the same id

NOT (

) $Y \neq Y'$ or $Z \neq Z'$

Catalog provides a unique price for each product (similar negative pattern)

Shorthand: variable repetitions instead of explicit equalities. All edges are labeled here by / (omitted)

Function specifications

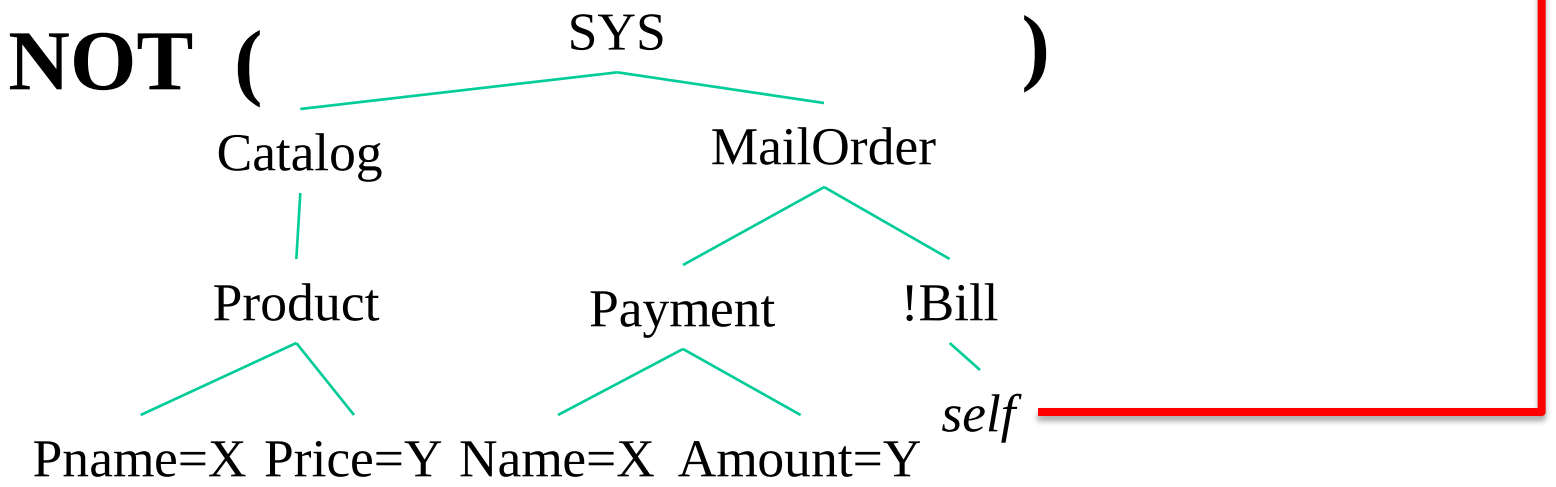
- All functions are external in this application

- !MailOrder
  - Kind: continuous
  - Call guard: true
  - Argument: empty
- We keep receiving MailOrders till the end of the world

!Bill

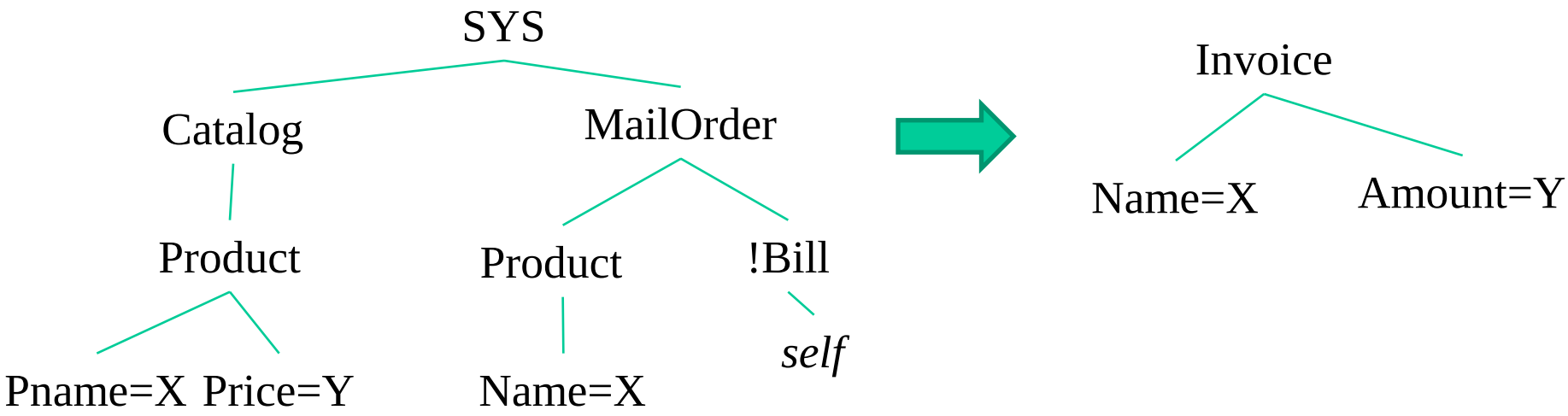
- !Bill
  - Kind: continuous
  - Call guard

Self binds to node labeled !Bill at which the call is made



The usefulness of the guard: Once the customer has paid the proper amount, the guard becomes negative and the function is disabled

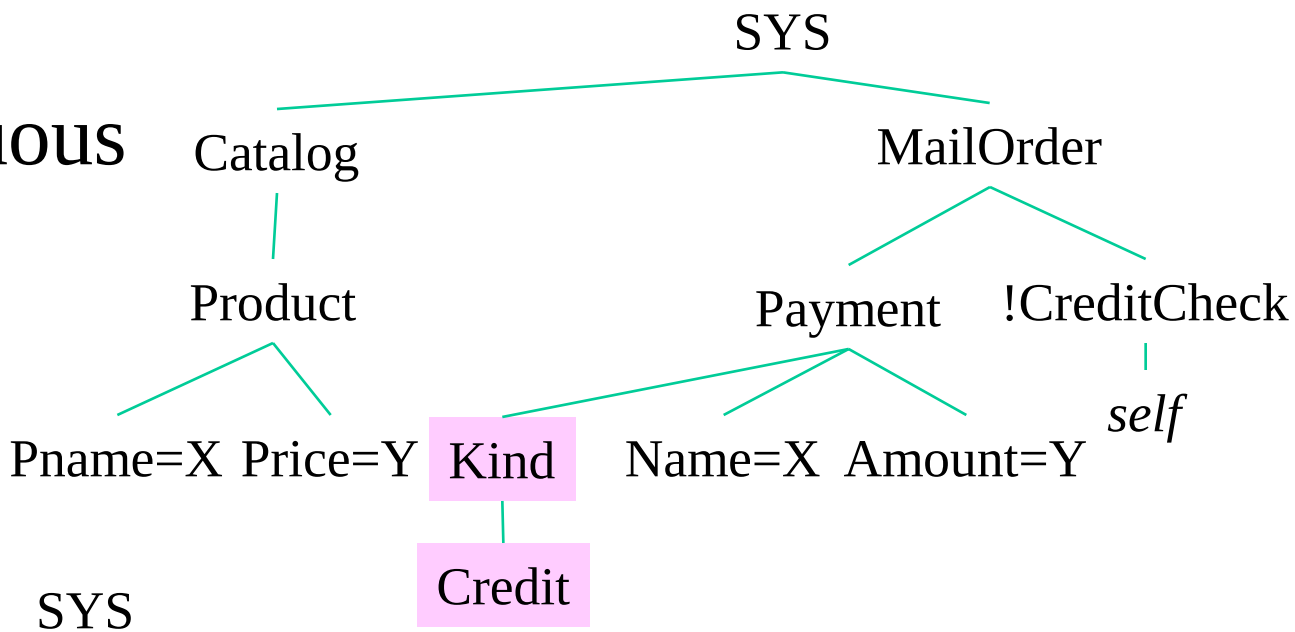
!Bill continued



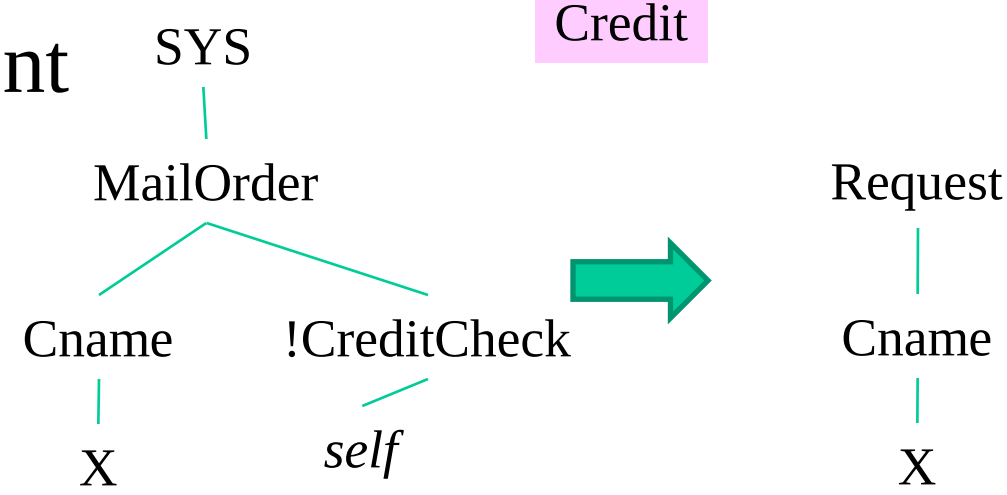
The query argument

!CreditCheck

- Noncontinuous
- Call guard



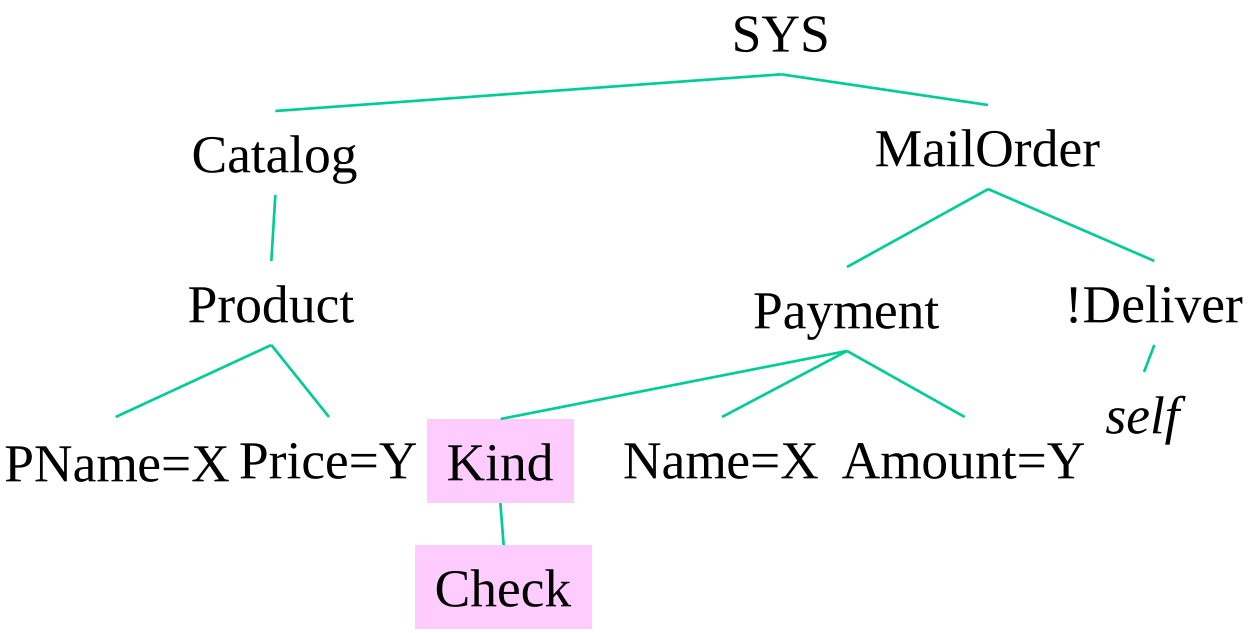
- Argument query



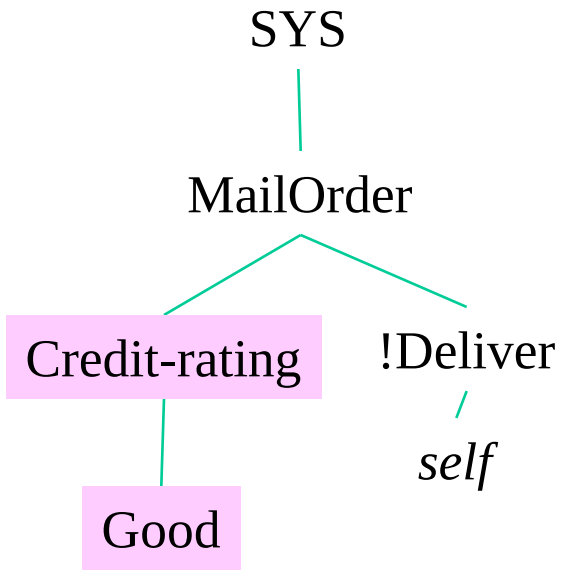
!Deliver

Kind: non-continuous

Call guard: Paid-check-ok OR Credit-ok



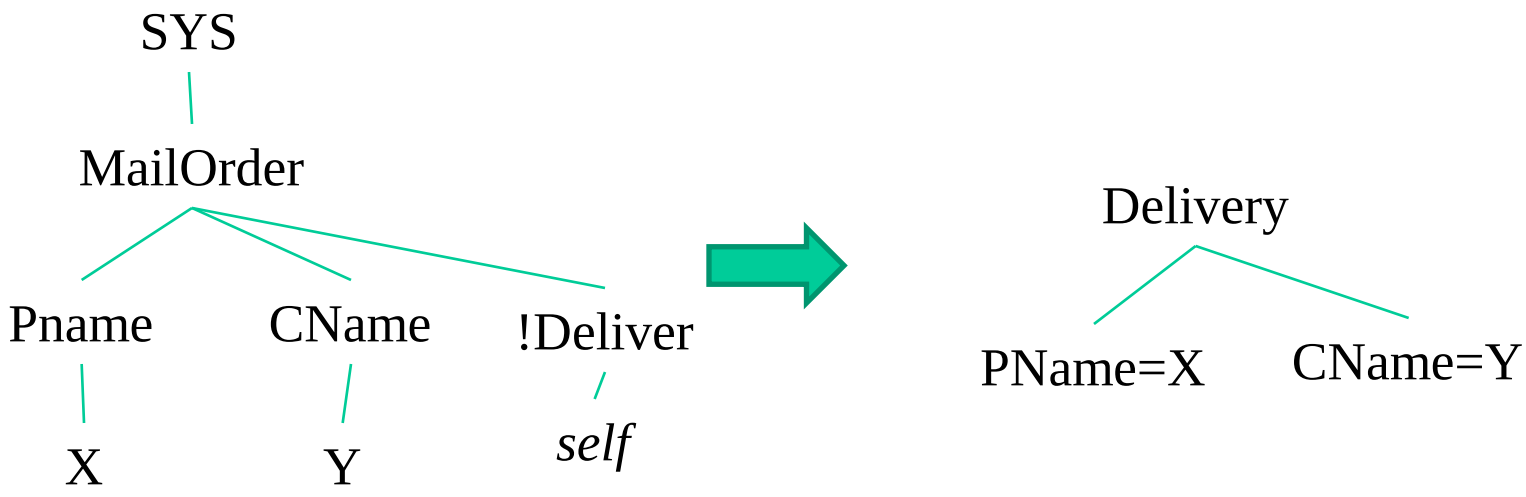
Paid-check-ok



Credit-ok

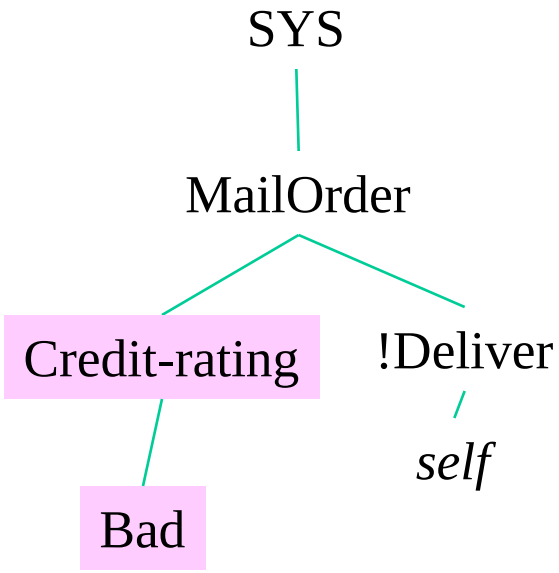
!Deliver - continued

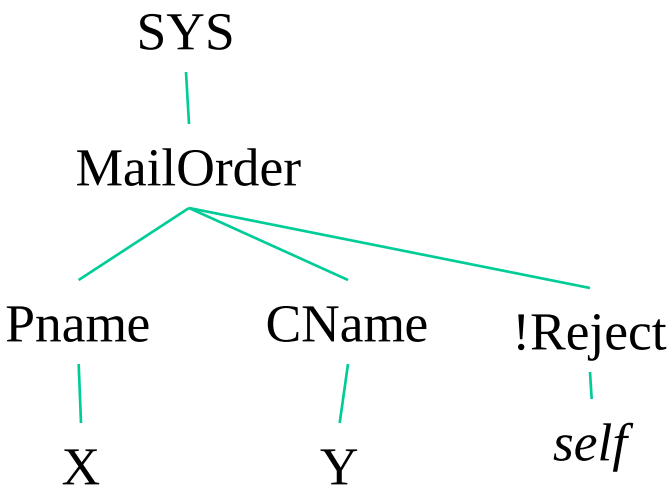
- Argument query

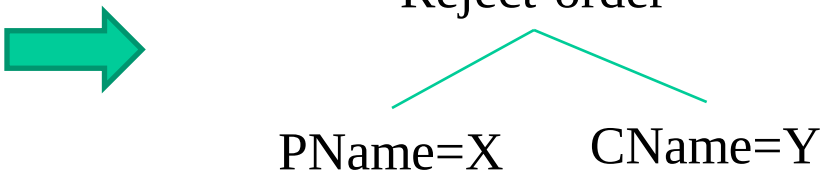


!Reject

- Kind: noncontinuous
- Call guard
- Arg query







Specification of temporal properties: Tree-LTL

- Boolean combinations of tree patterns and LTL operators
- Syntax of Tree-LTL

  φ :- pattern | φ and φ | φ or φ | not φ | φ **U** φ | **X** φ
- pattern(X1,…,Xn) : all other variables are seen as existentially quantified
- **X**: next **U**: until
  - Also **G**: always, **F**: eventually, etc
- Tree-LTL sentence ∀ φ
  - All free variables are *quantified universally at the end*
  - These are all the free variables from patterns

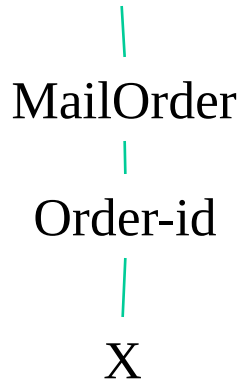
Example

Every mail order is eventually completed (delivered or rejected)

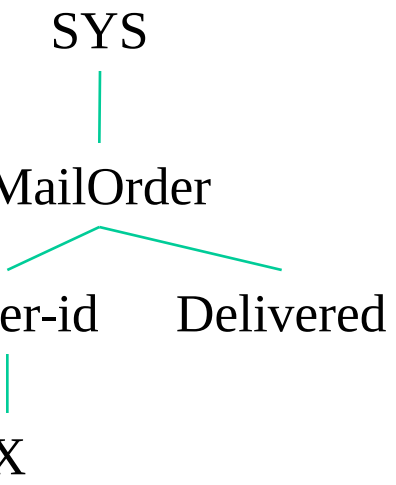
$\forall$ X [**G** ((T_1(X) → **F** (T_2 (X) $\vee$ T3 (X)))]

where

T1(X):



T2(X):



T3(X) : like T2(X) with Rejected instead of Delivered

This property is **false** the customer may forever pay the wrong amount

Semantics

- Natural extension of LTL, where each tree pattern is interpreted as a proposition in each instance of the run
- A run satisfies a Tree-LTL sentence $\forall X \varphi(X)$ if it satisfies $\varphi(h(X))$ for every valuation h of X into the set of data values used in the run

  Here $\varphi(h(X))$ is obtained by replacing in all patterns, each free occurrence of a variable Y in X by the data value $h(Y)$
- A service satisfies a Tree-LTL sentence $\forall X \varphi(X)$

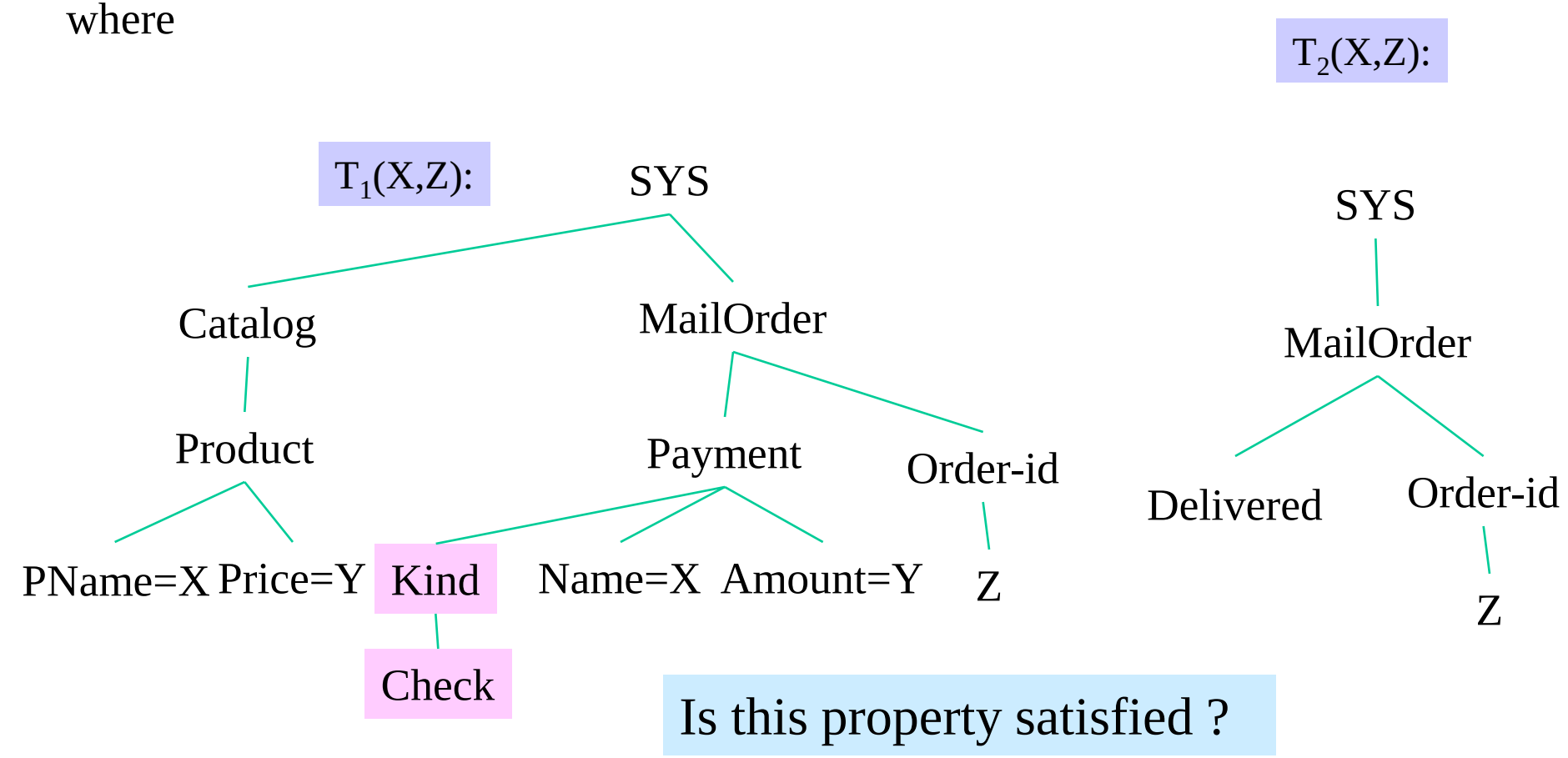
  if every run of the service satisfies it

Example – continued

Every product for which a correct payment by check has been made is eventually delivered

$\forall X \ \forall Z [\ \mathbf{G} (\ (T_1(X, \ Z) \rightarrow \ \mathbf{F} \ (T_2 (X, Z))) \ ]$

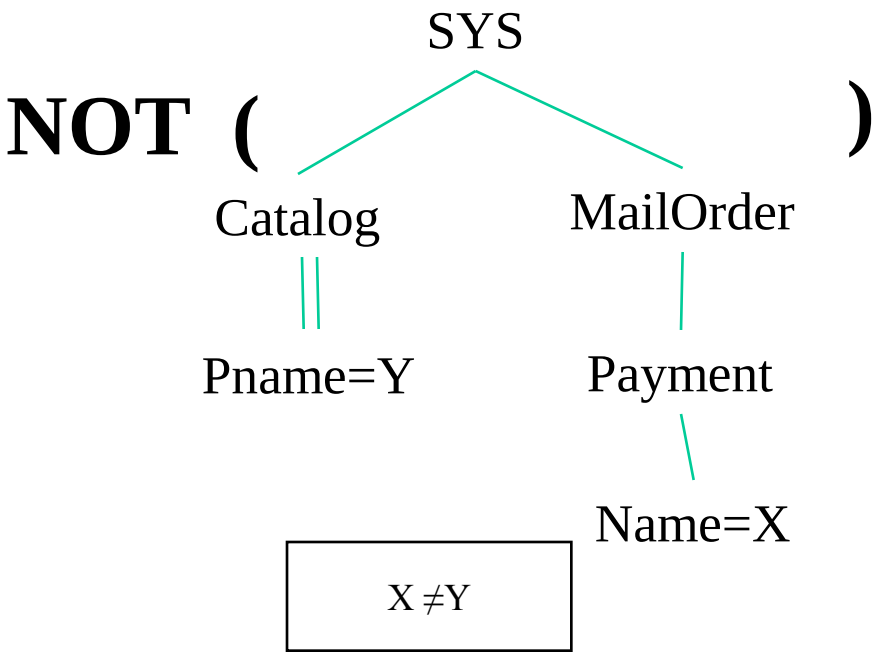
where



Is this property satisfied ?

Answer: NO! There is a bug in the spec: it allows the customer to pay the correct amount for a different product than the one initially ordered and eventually delivered !

Note: this can be fixed by adding as a data constraint



Notation for //

Other properties expressible in Tree-LTL

- No mail order can be rejected and later delivered (true)
- A check payment may never be made after a credit card payment attempt for the same order (false)
- A rejected order is blocking (no further function calls can be made) (true)
- An order can be rejected only if a payment attempt by credit card was made (true)

Other properties reducible to Tree-LTL sentences

- Confluence
- Termination

  all runs eventually reach a blocking instance
- Successful termination

  all runs eventually reach a blocking instance

  with no running calls

Results

- Focus: deciding whether a service S satisfies a Tree-LTL sentence
- Decidable for *bounded* services ☺

  syntactic restriction ensuring that all runs reach a blocking instance after a constant number of function calls (basically acyclicity of function calls)
- Undecidable as soon as any of the syntactic restrictions are relaxed ☹
  - Very careful analysis to obtain decidability (LS-VV)

Relaxations that lead to undecidability

- Continuous functions
- Arbitrary number of occurrences of !f in correct trees
- Arbitrary services even in absence of data
- Cyclic call graph (even with non continuous functions)
- Tree patterns with negated subpatterns
- Branching temporal quantifiers

Corollary

The following are decidable for bounded services:

- successful termination
  every run eventually reaches a blocking instance with no running calls
- confluence
  all runs from the same initial instance reach the same final instance

merci