# Formal Islands and Certified Pattern Matching Code

Claude Kirchner

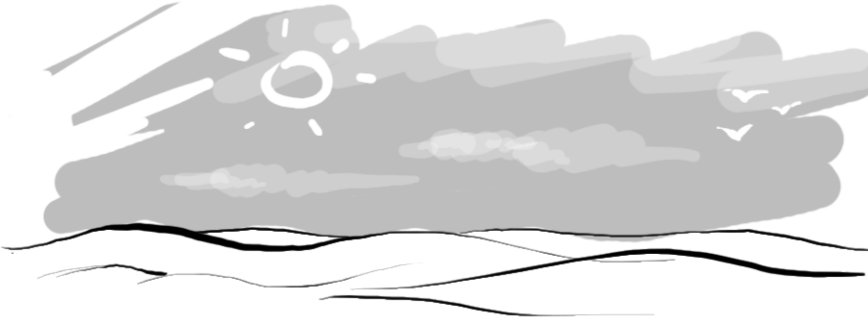
joint work with

Pierre-Étienne Moreau, Antoine Reilles

INRIA & LORIA & CNRS

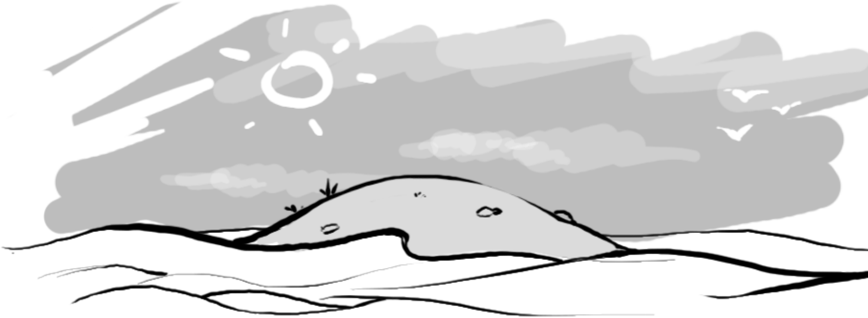
September 7, 2005

# Principle schema, 1



Existing code
JAVA program

# Principle schema, 2



Algebraic anchorage
Links Java with the basic extensions

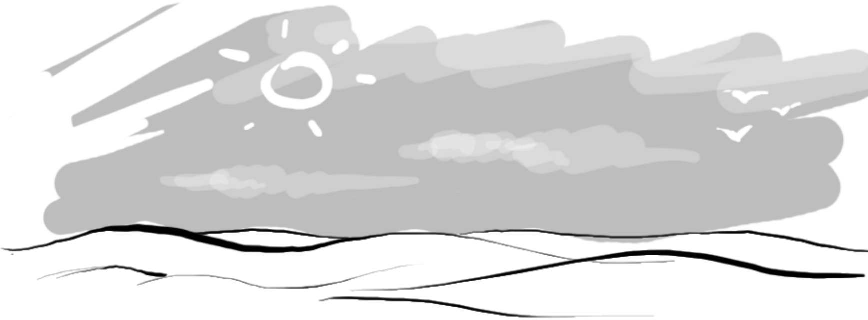
# Principle schema, 3

Building of the island
G → D

# Principle schema, 4

Building Validations
Property proofs

# Principle schema, end



Island dissolution
JAVA Program
Situation similar to the starting one, but we are sure of the code generated, thanks to the formal island.

# Formal Islands for Language Extension

- We use the notion of *Formal Island* and *anchoring* to extend an existing languages with formal capabilities
  - Anchoring means to describe new features in terms of the available functionalities of the host language
  - Once compiled, these features are translated into pure host language constructs

# Formal Islands Capabilities

1. To extend the expressivity of the language with higher-level constructs at design time
2. To perform formal proofs or transformations on the formal island constructions
3. To certify the implementation of the formal island compilation into the host language

# TOM

Allows for the formal island creation based on:

- matching
- rewriting
- traversals and strategies

In Java: JTOM
In C: CTOM
In CAML: CamTOM

`http://tom.loria.fr`

# The Tom language

Rewriting as a programming language:

$$
\begin{aligned}
fib(3) \rightarrow\ & fib(2) + fib(1) \\
& fib(2) + fib(1) \rightarrow\ fib(2) + 1 \\
& \qquad fib(2) + 1 \rightarrow\ fib(1) + fib(0) + 1 \\
& \qquad\qquad fib(1) + fib(0) + 1 \rightarrow\ \ldots
\end{aligned}
$$

# The Tom language

Rewriting as a programming language:

$$
\begin{aligned}
fib(3) \rightarrow\ & fib(2) + fib(1) \\
& fib(2) + fib(1) \rightarrow\ fib(2) + 1 \\
& \qquad fib(2) + 1 \rightarrow\ fib(1) + fib(0) + 1 \\
& \qquad\qquad fib(1) + fib(0) + 1 \rightarrow\ \ldots
\end{aligned}
$$

Tom: pattern matching compiler for Java, C, ...

# The Tom language

Rewriting as a programming language:

$$
\begin{aligned}
fib(3) \rightarrow\ & fib(2) + fib(1) \\
& fib(2) + fib(1) \rightarrow\ fib(2) + 1 \\
& \qquad fib(2) + 1 \rightarrow\ fib(1) + fib(0) + 1 \\
& \qquad\qquad fib(1) + fib(0) + 1 \rightarrow\ \ldots
\end{aligned}
$$

Tom: pattern matching compiler for Java, C, ...

| Rewriting languages |
|---|
| Fixed data-structure |
| Everything is rewriting |

# The Tom language

Rewriting as a programming language:

$$
\begin{aligned}
fib(3) \rightarrow\ & fib(2) + fib(1) \\
& fib(2) + fib(1) \rightarrow\ fib(2) + 1 \\
& \qquad fib(2) + 1 \rightarrow\ fib(1) + fib(0) + 1 \\
& \qquad\qquad fib(1) + fib(0) + 1 \rightarrow\ \ldots
\end{aligned}
$$

Tom: pattern matching compiler for Java, C, ...

| Rewriting languages | Tom |
|---|---|
| Fixed data-structure | plug-in (Objects, XML,...) |
| Everything is rewriting | Java, C,... |

# The Tom language

Rewriting as a programming language:

$$
\begin{array}{llll}
fib(3) \rightarrow & fib(2) + fib(1) & & \\
 & fib(2) + fib(1) \rightarrow & fib(2) + 1 & \\
 & & fib(2) + 1 \rightarrow & fib(1) + fib(0) + 1 \\
 & & & fib(1) + fib(0) + 1 \rightarrow \quad \ldots
\end{array}
$$

Tom: pattern matching compiler for Java, C, ...

```
fib(x) {
  %match(x) {
    zero()       ->{return `suc(zero());}
    suc(zero())->{return `suc(zero());}
    suc(suc(n))->{return `plus(fib(suc(n)),fib(n));}
  }
}
```

# The Tom language

Rewriting as a programming language:

$$
\begin{aligned}
fib(3) \rightarrow\ & fib(2) + fib(1) \\
& fib(2) + fib(1) \rightarrow\ fib(2) + 1 \\
& \qquad fib(2) + 1 \rightarrow\ fib(1) + fib(0) + 1 \\
& \qquad\qquad fib(1) + fib(0) + 1 \rightarrow\ \ldots
\end{aligned}
$$

Tom: pattern matching compiler for Java, C, ...

```
fib(x) {
  %match(x) {
    zero()       ->{return `suc(zero());}
    suc(zero())->{return `suc(zero());}
    suc(suc(n))->{return `plus(fib(suc(n)),fib(n));}
  }
}
```

Match against Java data-structures

# Pattern matching

**Typical example:**
The pattern $x + y$ matches the subject $1 + 2$ using the variable assignment $x \mapsto 1, y \mapsto 2$.

**In general, given:**
a pattern $p$
a subject $t$
find variables assignment $\sigma$ such that
$\sigma(p) = t$
Such a pattern matching problem is denoted $p \ll t$

# Objectives

- Need for certification of matching

# Objectives

- Need for certification of matching
- Independent of the used data-structures

# Objectives

- Need for certification of matching
- Independent of the used data-structures
- Compilation:

$$p \ll \quad \xrightarrow{compile} \quad P_p$$

# Objectives

- Need for certification of matching
- Independent of the used data-structures
- Compilation:

$$
\begin{array}{ccc}
p \ll t & \xrightarrow{\textit{compile}} & P_p(t) \\
\Big\downarrow \textit{match} & & \Big\downarrow \textit{eval} \\
\sigma & \xleftarrow[\textit{abstract}]{} & \epsilon
\end{array}
$$

# Objectives

- Need for certification of matching
- Independent of the used data-structures
- Compilation:

$$\begin{array}{ccc} p \ll t & \xrightarrow{\textit{compile}} & P_p(t) \\ \downarrow \textit{match} & & \downarrow \textit{eval} \\ \sigma & \xleftarrow[\textit{abstract}]{} & \epsilon \end{array}$$

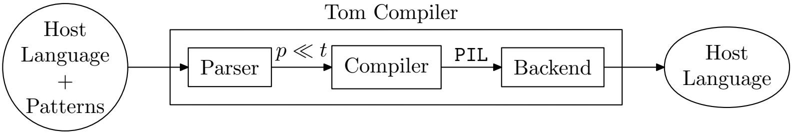
- Independent of the compilation process

# Quote

When we are using a compiler, we believe it is correct
When writing a compiler, we know it is incorrect!

*PEM*

# In the Tom compiler



# The *PIL* language

$$\textbf{instr} ::= \texttt{accept} \mid \texttt{refuse}$$

# The *PIL* language

$$
\begin{array}{lcl}
\textbf{instr} & ::= & \texttt{accept} \\
 & \mid & \texttt{refuse} \\
 & \mid & \texttt{if}(\textbf{expr}, \textbf{instr}, \textbf{instr}) \\
 & \mid & \texttt{let}(\ulcorner x \urcorner, \textbf{term}, \textbf{instr}) \quad (x \in \mathcal{X})
\end{array}
$$

# The *PIL* language

$$\begin{array}{lcl} \textbf{instr} & ::= & \texttt{accept} \\ & \mid & \texttt{refuse} \\ & \mid & \texttt{if}(\textbf{expr}, \textbf{instr}, \textbf{instr}) \\ & \mid & \texttt{let}(\ulcorner x \urcorner, \textbf{term}, \textbf{instr}) \quad (x \in \mathcal{X}) \end{array}$$

# The *PIL* language

$$
\begin{array}{lll}
\textbf{instr} & ::= & \texttt{accept} \\
 & \mid & \texttt{refuse} \\
 & \mid & \texttt{if}(\textbf{expr}, \textbf{instr}, \textbf{instr}) \\
 & \mid & \texttt{let}(\ulcorner x \urcorner, \textbf{term}, \textbf{instr}) \quad (x \in \mathcal{X})
\end{array}
$$

# The *PIL* language

$$
\begin{array}{lll}
\textbf{instr} & ::= & \texttt{accept} \\
 & \mid & \texttt{refuse} \\
 & \mid & \texttt{if}(\textbf{expr}, \textbf{instr}, \textbf{instr}) \\
 & \mid & \texttt{let}(\ulcorner x \urcorner, \textbf{term}, \textbf{instr}) \quad (x \in \mathcal{X}) \\
\textbf{expr} & ::= & \textbf{true} \mid \textbf{false} \\
 & \mid & \texttt{is\_fsym}(\textbf{term}, \ulcorner f \urcorner) \ (f \in \mathcal{F}) \\
 & \mid & \texttt{eq}(\textbf{term}, \textbf{term})
\end{array}
$$

# The *PIL* language

$$
\begin{array}{lll}
\textbf{instr} & ::= & \texttt{accept} \\
 & | & \texttt{refuse} \\
 & | & \texttt{if}(\textbf{expr}, \textbf{instr}, \textbf{instr}) \\
 & | & \texttt{let}(\ulcorner x \urcorner, \textbf{term}, \textbf{instr}) \quad (x \in \mathcal{X}) \\
\textbf{expr} & ::= & \textbf{true} \mid \textbf{false} \\
 & | & \texttt{is\_fsym}(\textbf{term}, \ulcorner f \urcorner) \ (f \in \mathcal{F}) \\
 & | & \texttt{eq}(\textbf{term}, \textbf{term}) \\
\textbf{term} & ::= & \ulcorner x \urcorner \ (x \in \mathcal{X}) \\
 & | & \ulcorner t \urcorner \ (t \in \mathcal{T}(\mathcal{F})) \\
 & | & \texttt{subterm}_f(\textbf{term}, \textbf{int}) \ (f \in \mathcal{F})
\end{array}
$$

# The *PIL* language

$$
\begin{array}{lll}
\textbf{instr} & ::= & \texttt{accept} \\
 & \mid & \texttt{refuse} \\
 & \mid & \texttt{if}(\textbf{expr}, \textbf{instr}, \textbf{instr}) \\
 & \mid & \texttt{let}(\ulcorner x \urcorner, \textbf{term}, \textbf{instr}) \quad (x \in \mathcal{X}) \\
\textbf{expr} & ::= & \textbf{true} \mid \textbf{false} \\
 & \mid & \texttt{is\_fsym}(\textbf{term}, \ulcorner f \urcorner) \ (f \in \mathcal{F}) \\
 & \mid & \texttt{eq}(\textbf{term}, \textbf{term}) \\
\textbf{term} & ::= & \ulcorner x \urcorner \ (x \in \mathcal{X}) \\
 & \mid & \ulcorner t \urcorner \ (t \in \mathcal{T}(\mathcal{F})) \\
 & \mid & \texttt{subterm}_f(\textbf{term}, \textbf{int}) \ (f \in \mathcal{F})
\end{array}
$$

Access the tree structure

# Definition of mapping : why and what ?

- Mapping between algebraic terms and their object representation
- How to represent a term (How to make it)?
- How to destruct a term?

# Definition of mapping : why and what ?

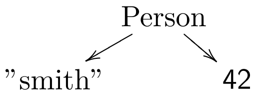
| Algebraic space | Concrete space |
| --- | --- |
| Abstract Data type<br>Person(name:String,age:**int**) | |

# Definition of mapping : why and what ?

Algebraic space | Concrete space

Abstract Data type

Person(name:String,age:**int**)

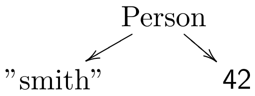


# Definition of mapping : why and what ?

## Algebraic space

### Abstract Data type

```
Person(name:String,age:int)
```




## Concrete space

```
class Person {
  String name;
  String firstName;
  int age;
}
```

# Definition of mapping : why and what ?

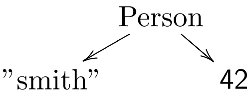
Algebraic space

Abstract Data type

Person(name:String,age:**int**)

Concrete space

```
class Person {
  String name;
  String firstName;
  int age;
}
```




| Person |
| --- |
| "smith" |
| "bob" |
| 42 |

# Definition of mapping : why and what ?

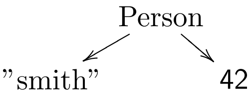
## Algebraic space

### Abstract Data type

Person(name:String,age:**int**)

## Concrete space

```
class Person {
  String name;
  String firstName;
  int age;
}
```




$Symb(t)$

$t_{|1}$

$t_{|2}$

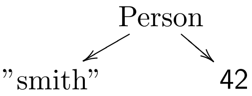
| Person |
|---|
| "smith" |
| "bob" |
| 42 |

# Definition of mapping : why and what ?

## Algebraic space

### Abstract Data type

Person(name:String,age:**int**)




$\mathcal{S}ymb(t)$

$t_{|1}$

$t_{|2}$

## Concrete space

```
class Person {
  String name;
  String firstName;
  int age;
}
```

| Person |
|---|
| "smith" |
| "bob" |
| 42 |

`instanceof`

`t.name`

`t.age`

# Definition of mapping : why and what ?

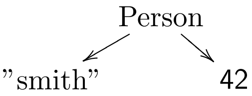
**Algebraic space**

Abstract Data type

Person(name:String,age:**int**)

**Concrete space**

```
class Person {
  String name;
  String firstName;
  int age;
}
```




$\mathcal{S}ymb(t)$

$t_{|1}$

$t_{|2}$

| Person |
|---|
| "smith" |
| "bob" |
| 42 |

`instanceof`

`t.name`

`t.age`

$$\texttt{is\_fsym}(\lceil t \rceil, \lceil f \rceil) \equiv \lceil \mathcal{S}ymb(t) = f \rceil$$

$$\texttt{subterm}_f(\lceil t \rceil, \lceil i \rceil) \equiv \lceil t_{|i} \rceil$$

# Validation method

Consider the pattern $g(f(x), b)$ and the term $g(f(a), b)$ and the program $P$

$$\begin{array}{ccc} g(f(x), b) \ll g(f(a), b) & \xrightarrow{\textit{compile}} & P(\lceil g(f(a), b) \rceil) \\ \textit{match} \downarrow & & \downarrow \textit{semantics} \\ [x \leftarrow a] & \xleftarrow[\textit{abstract}]{} & \{x := \lceil a \rceil\} \end{array}$$

# Validation method

Consider the pattern $g(f(x), b)$ and the term $g(f(a), b)$ and the program $P$

$$\begin{array}{ccc} g(f(x), b) \ll X & \xrightarrow{\textit{compile}} & P(\lceil X \rceil) \\ \textit{match} \downarrow & & \downarrow \textit{semantics} \\ [x \leftarrow X_{|1_{|1}}] & \xleftarrow[\textit{abstract}]{} & \{x := \lceil X_{|1_{|1}} \rceil\} \end{array}$$

# Big-step semantics

Rules for instructions **instr**:

$$\frac{}{\langle \epsilon, \texttt{accept} \rangle \mapsto \langle \epsilon, \texttt{accept} \rangle} \quad (\textit{accept})$$

$$\frac{}{\langle \epsilon, \texttt{refuse} \rangle \mapsto \langle \epsilon, \texttt{refuse} \rangle} \quad (\textit{refuse})$$

# Big-step semantics

Rules for instructions **instr**:

$$\frac{}{\langle \epsilon, \texttt{accept} \rangle \mapsto \langle \epsilon, \texttt{accept} \rangle} \quad (\textit{accept})$$

$$\frac{}{\langle \epsilon, \texttt{refuse} \rangle \mapsto \langle \epsilon, \texttt{refuse} \rangle} \quad (\textit{refuse})$$

$$\frac{\langle \epsilon, i_1 \rangle \mapsto \langle \epsilon', i \rangle}{\langle \epsilon, \texttt{if}(e, i_1, i_2) \rangle \mapsto \langle \epsilon', i \rangle} \text{ if } \epsilon(e) \equiv \texttt{true} \quad (\textit{iftrue})$$

# Big-step semantics

Rules for instructions **instr**:

$$\frac{}{\langle \epsilon, \texttt{accept} \rangle \mapsto \langle \epsilon, \texttt{accept} \rangle} \qquad (\textit{accept})$$

$$\frac{}{\langle \epsilon, \texttt{refuse} \rangle \mapsto \langle \epsilon, \texttt{refuse} \rangle} \qquad (\textit{refuse})$$

$$\frac{\langle \epsilon, i_1 \rangle \mapsto \langle \epsilon', i \rangle}{\langle \epsilon, \texttt{if}(e, i_1, i_2) \rangle \mapsto \langle \epsilon', i \rangle} \text{ if } \epsilon(e) \equiv \texttt{true} \qquad (\textit{iftrue})$$

$$\frac{\langle \epsilon, i_2 \rangle \mapsto \langle \epsilon', i \rangle}{\langle \epsilon, \texttt{if}(e, i_1, i_2) \rangle \mapsto \langle \epsilon', i \rangle} \text{ if } \epsilon(e) \equiv \texttt{false} \qquad (\textit{iffalse})$$

# Big-step semantics

Rules for instructions **instr**:

$$\frac{}{\langle \epsilon, \texttt{accept} \rangle \mapsto \langle \epsilon, \texttt{accept} \rangle} \qquad (accept)$$

$$\frac{}{\langle \epsilon, \texttt{refuse} \rangle \mapsto \langle \epsilon, \texttt{refuse} \rangle} \qquad (refuse)$$

$$\frac{\langle \epsilon, i_1 \rangle \mapsto \langle \epsilon', i \rangle}{\langle \epsilon, \texttt{if}(e, i_1, i_2) \rangle \mapsto \langle \epsilon', i \rangle} \text{ if } \epsilon(e) \equiv \texttt{true} \qquad (iftrue)$$

$$\frac{\langle \epsilon, i_2 \rangle \mapsto \langle \epsilon', i \rangle}{\langle \epsilon, \texttt{if}(e, i_1, i_2) \rangle \mapsto \langle \epsilon', i \rangle} \text{ if } \epsilon(e) \equiv \texttt{false} \qquad (iffalse)$$

$$\frac{\langle \epsilon[x \leftarrow \lceil t \rceil], i_1 \rangle \mapsto \langle \epsilon', i \rangle}{\langle \epsilon, \texttt{let}(x, u, i_1) \rangle \mapsto \langle \epsilon', i \rangle} \text{ if } \epsilon(u) \equiv \lceil t \rceil \qquad (let)$$

# Definition of a correct compilation

## Definition

Given a formal anchor $\lceil\rceil$, a well-formed program $\pi_p$ is a *correct* compilation of $p$ when both:

If the program results in accept, then a match is computed
$\forall \epsilon, \epsilon' \in \mathcal{E}nv, \forall t \in \mathcal{T}(\mathcal{F})$,

$$\langle \epsilon, \pi_p(\lceil t \rceil) \rangle \mapsto \langle \epsilon', \texttt{accept} \rangle \Rightarrow \Phi(\epsilon')(p) = t \qquad (sound_{OK})$$

If $p$ matches $t$, then the program finds a match
$\forall \epsilon \in \mathcal{E}nv, \forall t \in \mathcal{T}(\mathcal{F})$,

$$p \ll t \Rightarrow \exists \epsilon' \in \mathcal{E}nv, \langle \epsilon, \pi_p(\lceil t \rceil) \rangle \mapsto \langle \epsilon', \texttt{accept} \rangle \wedge \Phi(\epsilon')(p) = t \qquad (complete_{OK})$$

# Result

## Theorem

*Given a formal anchor $\lceil\rceil$, a pattern $p \in \mathcal{T}(\mathcal{F},\mathcal{X})$, and a well-formed program $\pi_p \in \texttt{PIL}$, we have:*

$$\pi_p \textit{ is a correct compilation of } p$$
$$\Longleftrightarrow$$
$$\forall \epsilon, \epsilon' \in \mathcal{E}nv, \forall t \in \mathcal{T}(\mathcal{F}),$$
$$\langle \epsilon, \pi_p(\lceil t \rceil) \rangle \mapsto \langle \epsilon', \texttt{accept} \rangle \Leftrightarrow \Phi(\epsilon')(p) = t$$

# Result

## Theorem

*Given a formal anchor $\lceil\rceil$, a pattern $p \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and a well-formed program $\pi_p \in \mathtt{PIL}$, we have:*

$$\pi_p \textit{ is a correct compilation of } p$$
$$\Longleftrightarrow$$
$$\forall \epsilon, \epsilon' \in \mathcal{E}nv, \forall t \in \mathcal{T}(\mathcal{F}),$$
$$\langle \epsilon, \pi_p(\lceil t \rceil) \rangle \mapsto \langle \epsilon', \mathtt{accept} \rangle \Leftrightarrow \Phi(\epsilon')(p) = t$$

Now, how to use these tools ?

# Method

Let's consider the pattern $p:\ g(f(x),b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\qquad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\qquad\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\qquad\qquad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\qquad\qquad\quad \mathtt{refuse})), \\
\qquad\quad \mathtt{refuse}), \\
\qquad \mathtt{refuse})
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\qquad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\qquad\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\qquad\qquad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\qquad\qquad\quad \mathtt{refuse})), \\
\qquad\quad \mathtt{refuse}), \\
\qquad \mathtt{refuse})
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true}$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true}$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true}$$
$$\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil) \equiv \mathtt{true}$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil) \equiv \mathtt{true}
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\qquad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\qquad\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\qquad\qquad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\qquad\qquad\quad \mathtt{refuse})), \\
\qquad\quad \mathtt{refuse}), \\
\qquad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil) \equiv \mathtt{true} \\
x_1 = \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1)
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil) \equiv \mathtt{true} \\
x_1 = \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1)
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil) \equiv \mathtt{true} \\
x_1 = \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1) \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil) \equiv \mathtt{true}
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{is\_fsym}(\mathtt{s}, \lceil g \rceil) \equiv \mathtt{true} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \lceil f \rceil) \equiv \mathtt{true} \\
x_1 = \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1) \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \lceil b \rceil) \equiv \mathtt{true}
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \ulcorner g \urcorner), \\
\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \ulcorner f \urcorner), \\
\qquad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\qquad\qquad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \ulcorner b \urcorner), \\
\qquad\qquad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\qquad\qquad\quad \mathtt{refuse})), \\
\qquad\quad \mathtt{refuse}), \\
\qquad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{is\_fsym}(\mathtt{s}, \ulcorner g \urcorner) \equiv \mathtt{true} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \ulcorner f \urcorner) \equiv \mathtt{true} \\
x_1 = \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1) \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \ulcorner b \urcorner) \equiv \mathtt{true} \\
x = x_1
\end{array}
$$

# Deriving constraints for $g(f(x), b)$

$$
\begin{array}{l}
\pi_{g(f(x),b)}(\mathtt{s}) \triangleq \\
\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{s}, \ulcorner g \urcorner), \\
\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \ulcorner f \urcorner), \\
\quad\quad\quad \mathtt{let}(x_1, \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1), \\
\quad\quad\quad\quad \mathtt{if}(\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \ulcorner b \urcorner), \\
\quad\quad\quad\quad\quad \mathtt{let}(x, x_1, \mathtt{accept}), \\
\quad\quad\quad\quad\quad \mathtt{refuse})), \\
\quad\quad\quad \mathtt{refuse}), \\
\quad\quad \mathtt{refuse})
\end{array}
$$

$$
\begin{array}{ll}
\mathtt{is\_fsym}(\mathtt{s}, \ulcorner g \urcorner) \equiv \mathtt{true} & \to \mathcal{S}ymb(\mathtt{s}) = g \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 1), \ulcorner f \urcorner) \equiv \mathtt{true} & \to \mathcal{S}ymb(\mathtt{s}_{|1}) = f \\
x_1 = \mathtt{subterm}_f(\mathtt{subterm}_g(\mathtt{s}, 1), 1) & \to x_1 = \mathtt{s}_{|1_{|1}} \\
\mathtt{is\_fsym}(\mathtt{subterm}_g(\mathtt{s}, 2), \ulcorner b \urcorner) \equiv \mathtt{true} & \to \mathcal{S}ymb(\mathtt{s}_{|2}) = b \\
x = x_1 & \to x = \mathtt{s}_{|1_{|1}}
\end{array}
$$

# Proof obligation

We have to prove:

$$\forall \mathtt{s}, x : g(f(x), b) = \mathtt{s} \Leftrightarrow$$

# Proof obligation

We have to prove:

$$\forall \mathrm{s}, x : g(f(x), b) = \mathrm{s} \Leftrightarrow \begin{array}{l} \;\; \mathcal{S}ymb(\mathrm{s}) = g \\ \wedge\, \mathcal{S}ymb(\mathrm{s}_{|1}) = f \\ \wedge\, \mathcal{S}ymb(\mathrm{s}_{|2}) = b \\ \wedge\, x = \mathrm{s}_{|1_{|1}} \end{array}$$

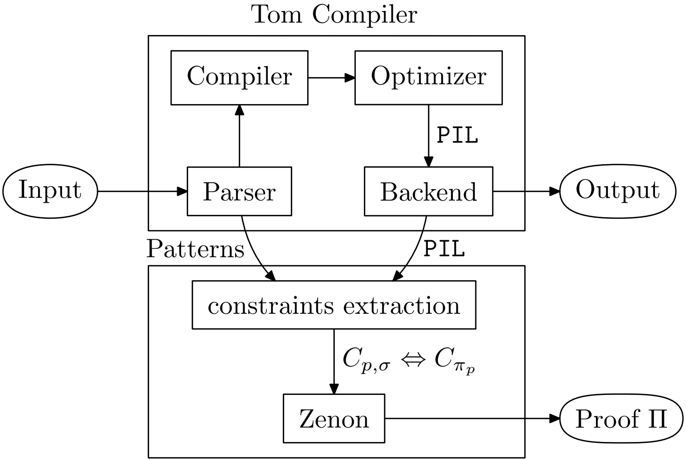
# Proof obligation

We have to prove:

$$\forall \mathrm{s}, x : g(f(x), b) = \mathrm{s} \Leftrightarrow \begin{array}{l} \mathcal{S}ymb(\mathrm{s}) = g \\ \wedge\, \mathcal{S}ymb(\mathrm{s}_{|1}) = f \\ \wedge\, \mathcal{S}ymb(\mathrm{s}_{|2}) = b \\ \wedge\, x = \mathrm{s}_{|1_{|1}} \end{array}$$

Handled by Zenon [Doligez, INRIA], first order theorem prover

# In the Tom compiler



# Application

Application on the Tom compiler itself

- more than 200 patterns
- verification takes about 20% of the compilation time

# Conclusion

- Contributions
  - Formal Island concept: safely build on existing code
  - Certification of pattern matching code
  - Model of the mapping between formal and actual data-structure
- Perspectives
  - Extention to rewriting (inlining)
  - Decidability
  - Extention to associative matching
  - Support for equational theories

`http://tom.loria.fr`