univaq lecture series:

Software Language Engineering

meets

Model-driven Engineering

Ralf Lämmel
Software Languages Team
University of Koblenz-Landau
http://www.softlang.org/

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

univaq lecture series

- Tue May 23, 9:30am–11:30am

Internal DSLs

- Wed May 24, 11:30am–1:30pm

External DSLs

- Tue Jun 6, 9:30am–11:30am

Metaprogramming

- Wed Jun 7, 11:30am–1:30pm

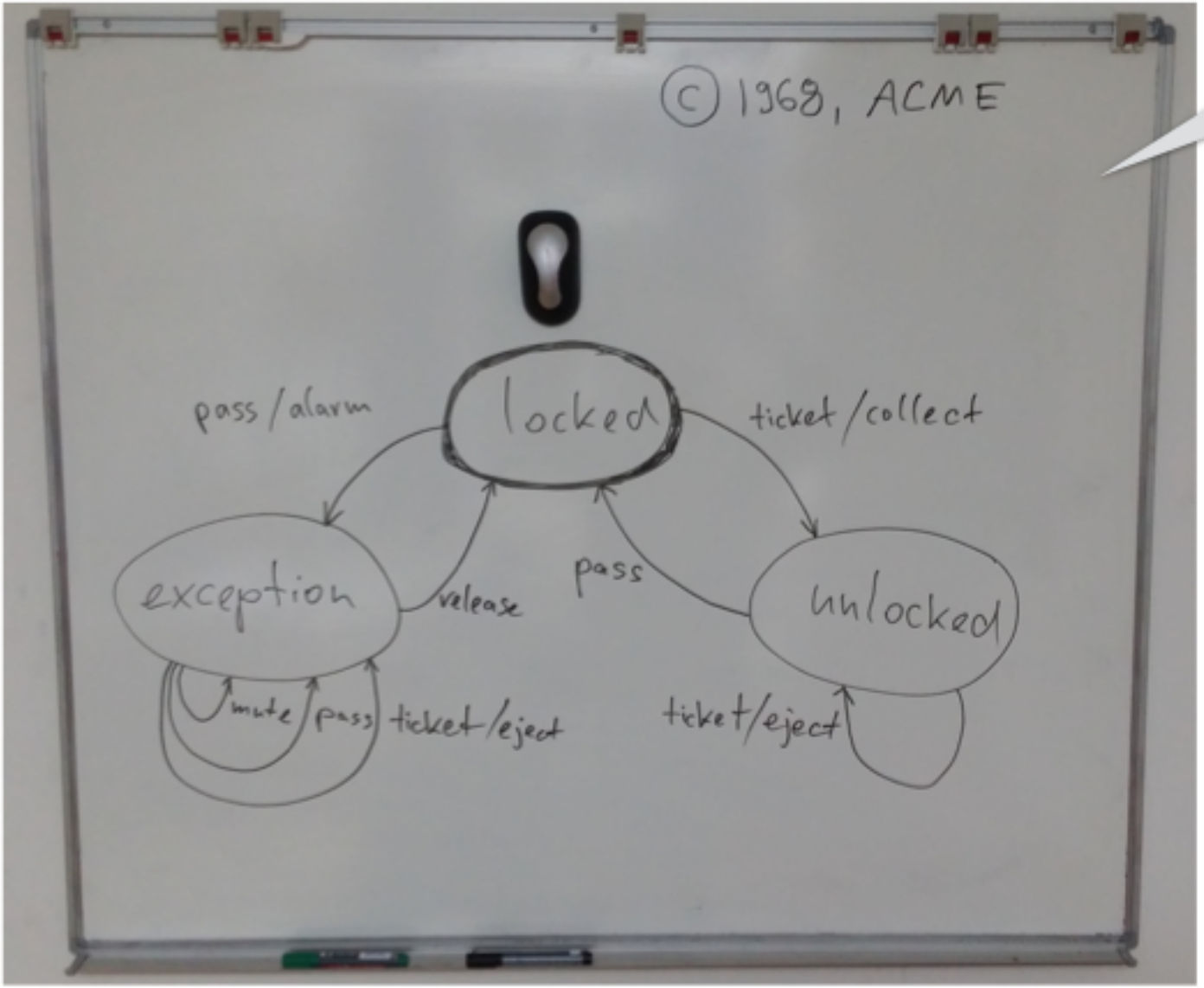
Megamodeling

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A quick introduction to domain-specific languages (DSLs)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A DSL for finite state machines (FSMs)

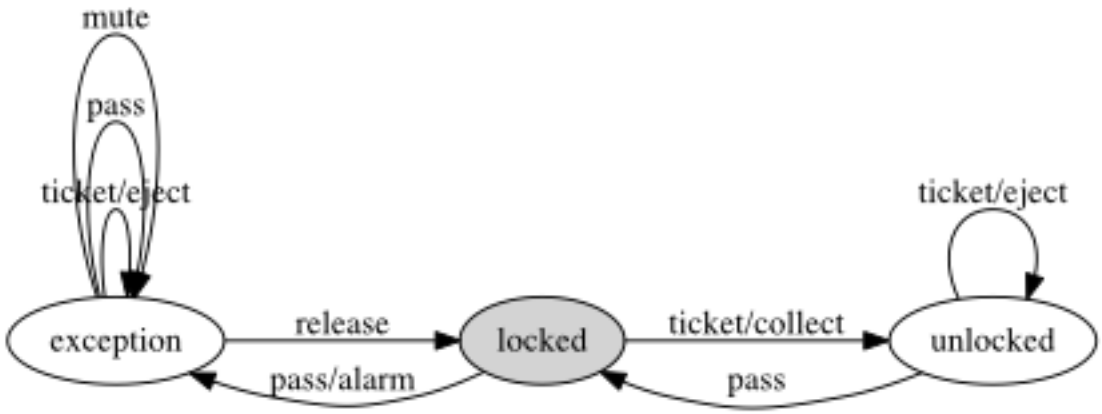


A FSM for a 'turnstile' in a metro system

Imagine the FSM language (FSML) to have started on the black/whiteboard a long time ago.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Concepts of FSML illustrated for the turnstile FSM



States

locked The turnstile is locked. No passenger is allowed to pass.
unlocked The turnstile is unlocked. A passenger may pass.
exception A problem has occurred and metro personnel needs to intervene.

Events

ticket A passenger inserts a ticket into the card reader.
pass A passenger passes the turnstile as noticed by a sensor.
mute Metro personnel turns off alarm after exception.
release Metro personnel turns on normal operation again.

e/a

Actions

collect The ticket is collected by the card reader.
eject The ticket is ejected by the card reader.
alarm An alarm is turned on and metro personnel is requested.

e/**a**

Transitions

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Semantics (I/O behavior) of FSML illustrated for the turnstile FSM

Input (= sequence of events)

ticket	*A ticket is inserted. (The turnstile is unlocked, thus.)*
ticket	*Another ticket is inserted. (The superflous ticket is ejected.)*
pass	*Someone passes the turnstile. (This is Ok.)*
pass	*Someone passes the turnstile. (This triggers alarm.)*

Output (= sequence of actions)

collect	*The inserted ticket is collected.*
eject	*A ticket inserted in unlocked state is ejected.*
alarm	*An attempt to pass in locked state triggers alarm.*

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

DSL implementation in different ‘styles’

- **External DSL**:

  Designated parser, checker, interpreter, compiler

- **Internal DSL**:

  Implementation as library using host language features

Our initial focus

N.B.: This is a gross oversimplification.
There are options or hybrids using extensible languages, extensible compilers, metaprogramming systems, and language workbenches.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Online resources

YAS' GitHub repository contains all code.

YAS (Yet Another SLR (Software Language Repository))

http://www.softlang.org/yas

See here specifically:

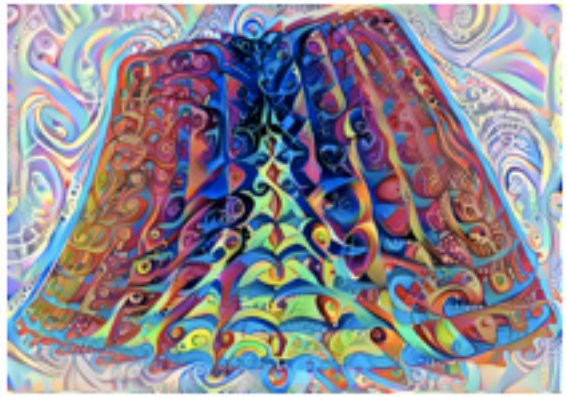
https://github.com/softlang/yas/tree/master/languages/FSML

Subdirectories Java and Python

The Software Languages Book

http://www.softlang.org/book

See Chapter 2 in particular.



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Internal DSL style

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

We are going to do here …

Internal DSL style

with

and

libraries

N.B.: If we were using C++, Scheme, Haskell, or others for internal DSL implementation, additional or different techniques could or should be leveraged, e.g., operator overloading, macros, or templates.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

```
turnstile = new Fsm();
State s = new State();
s.setStateid("locked");
s.setInitial(true);
turnstile.getStates().add(s);
s = new State();
s.setStateid("unlocked");
turnstile.getStates().add(s);
s = new State();
s.setStateid("exception");
turnstile.getStates().add(s);
Transition t = new Transition();
t.setSource("locked");
t.setEvent("ticket");
t.setAction("collect");
t.setTarget("unlocked");
turnstile.getTransitions().add(t);
t = new Transition();
... add more transitions ...
```

‘Imperative’ Java API

N.B.: Arguably this was common style until perhaps 200?. Even today, this sort of code may still be written.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Java API with functional constructors

```
turnstile = new Fsm();
turnstile.getStates().add(new State("locked", true));
turnstile.getStates().add(new State("unlocked"));
turnstile.getStates().add(new State("exception"));
turnstile.getTransitions().add(new Transition("locked", "ticket", "collect", "unlocked"));
turnstile.getTransitions().add(new Transition("locked", "pass", "alarm", "exception"));
... add more transitions ...
```

N.B.: Functional constructors have been used by C++ and Java et al. programmers for a long time, but they are insufficient to avoid repetitive code and to hide the internal representation.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Basic Java object model including functional constructors

```
public class Fsm {
   private List<State> states = new LinkedList<>();
   private List<Transition> transitions = new LinkedList<>();
   public List<State> getStates() { return states; }
   public List<Transition> getTransitions() { return transitions; }
}
public class State {
   private String id;
   private boolean initial;
   public String getStateid() { return id; }
   public void setStateid(String state) { this.id = state; }
   public boolean isInitial() { return initial; }
   public void setInitial(boolean initial) { this.initial = initial; }
   public State() { }
   public State(String id) { this.id = id; }
   public State(String id, boolean initial) { this.id = id; this.initial = initial; }
}
public class Transition {
   private String source;
   private String event;
   private String action;
```

It's easy,

but not what we want.

```
  private String id;
  private boolean initial;
  public String getStateid() { return id; }
  public void setStateid(String state) { this.id = state; }
  public boolean isInitial() { return initial; }
  public void setInitial(boolean initial) { this.initial = initial; }
  public State() { }
  public State(String id) { this.id = id; }
  public State(String id, boolean initial) { this.id = id; this.initial = initial; }
}
public class Transition {
  private String source;
  private String event;
  private String action;
  private String target;
  ... getters and setters ...
  public Transition() { }
  public Transition(String source, String event, String action, String target) {
    this.source = source;
    this.event = event;
    this.action = action;
    this.target = target;
  }
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Use of a **fluent API** in Java

```
Fsm turnstile =
  fsm()
    .addState("locked")
      .addTransition("ticket", "collect", "unlocked")
      .addTransition("pass", "alarm", "exception")
    .addState("unlocked")
      .addTransition("ticket", "eject", "unlocked")
      .addTransition("pass", null, "locked")
    .addState("exception")
      .addTransition("ticket", "eject", "exception")
      .addTransition("pass", null, "exception")
      .addTransition("mute", null, "exception")
      .addTransition("release", null, "locked");
```

Leveraged techniques:

- Factory methods
- Method chaining
- Implicit parameters
- Conventions (defaults)

N.B.: The current state is maintained along the way.

The state declared first is assumed to be the initial one.

The representation is not revealed—no constructors are used.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Use of a **fluent API** in Python

```
turnstile = Fsm() \
    .addState("locked") \
        .addTransition("ticket", "collect", "unlocked") \
        .addTransition("pass", "alarm", "exception") \
    .addState("unlocked") \
        .addTransition("ticket", "eject", "unlocked") \
        .addTransition("pass", None, "locked") \
    .addState("exception") \
        .addTransition("ticket", "eject", "exception") \
        .addTransition("pass", None, "exception") \
        .addTransition("mute", None, "exception") \
        .addTransition("release", None, "locked")
```

Leveraged techniques:

- Factory methods
- Method chaining
- Implicit parameters
- Conventions (defaults)

N.B.: If we were using C++, Scheme, Haskell, or others for internal DSL implementation, additional or different techniques could or should be leveraged, e.g., operator overloading, macros, or templates.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Definition of **fluent API** in Java

```
public interface Fsm {
  public Fsm addState(String state);
  public Fsm addTransition(String event, String action, String target);
  public String getInitial();
  public ActionStatePair makeTransition(String state, String event);
}
```

Interface

```
public class ActionStatePair {
  public String action;
  public String state;
}
```

Helper type

N.B.: This interface does not expose the internal representation.
The interface does not just cover fluent construction;
it also covers 'observation' of the opaque representation.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Implementation of **fluent API** in Java

```
public class FsmImpl implements Fsm {
  private String initial; // the initial state
  private String current; // the "current" state
  // A cascaded map for maintaining states and transitions
  private HashMap<String, HashMap<String, ActionStatePair>> fsm =
      new HashMap<>();
  private FsmImpl() { }
  // Construct FSM object
  public static Fsm fsm() { return new FsmImpl(); }
  // Add state and set it as current state
  public Fsm addState(String id) {
    // First state is initial state
    if (initial == null) initial = id;
    // Remember state for subsequent transitions
    this.current = id;
    if (fsm.containsKey(id)) throw new FsmIDistinctIdsException();
    fsm.put(id, new HashMap<String, ActionStatePair>());
    return this;
  }
  // Add transition for current state
  public Fsm addTransition(String event, String action, String target) { ...
```

```
    if (fsm.containsKey(id)) throw new FsmlDistinctIdsException();
    fsm.put(id, new HashMap<String, ActionStatePair>());
    return this;
  }
  // Add transition for current state
  public Fsm addTransition(String event, String action, String target) {
    if (fsm.get(current).containsKey(event)) throw new FsmlDeterminismException();
    ActionStatePair pair = new ActionStatePair();
    pair.action = action;
    pair.state = target;
    fsm.get(current).put(event, pair);
    return this;
  }
  // Getter for initial state
  public String getInitial() {
    return initial;
  }
  // Make transition
  public ActionStatePair makeTransition(String state, String event) {
    if (!fsm.containsKey(state)) throw new FsmlResolutionException();
    if (!fsm.get(state).containsKey(event)) throw new FsmlInfeasibleEventException();
    return fsm.get(state).get(event);
  }
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A *JUnit* test case for simulation

```
public class FluentTest {

  private static final String[] input =
    {"ticket", "ticket", "pass", "pass", "ticket", "mute", "release"};
  private static final String[] output =
    {"collect", "eject", "alarm", "eject"};

  @Test
  public void runSample() {
    assertArrayEquals(output, run(Sample.turnstile, input));
  }
}
```

Sample input

Expected output

FSML interpreter

N.B.: This is how a Java programmer (a DSL user) would document a use case of a specific FSM (and validate intuitions).

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An interpreter (a 'simulator') in Java

```
public class FsmInterpreter {
  public static String[] run(Fsm fsm, String[] input) {
    ArrayList<String> output = new ArrayList<>();
    String state = fsm.getInitial();
    for (String event : input) {
      ActionStatePair pair = fsm.makeTransition(state, event);
      if (pair.action != null) output.add(pair.action);
      state = pair.state;
    }
    return output.toArray(new String[output.size()]);
  }
}
```

N.B.: The interpreter essentially models the dynamic semantics of FSML. This is a non-interactive interpreter. In practice, an interactive DSL implementation may be required.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Implementation of **fluent API** in Python

```
class Fsm():
    def __init__(self):
      self.fsm = defaultdict(list)
      self.current = None
    def addState(self, id):
        return self.addStateNoDefault(self.current is None, id)
    def addStateNoDefault(self, initial, id):
        if id in self.fsm[id]: raise FsmIDistinctIdsException;
        self.stateObject = dict()
        self.stateObject['transitions'] = defaultdict(list)
        self.stateObject['initial'] = initial
        self.fsm[id] += [self.stateObject]
        self.current = id
        return self
    def addTransition(self, event, action, target):
        if event in self.stateObject['transitions']: raise FsmIDeterminismException;
        self.stateObject['transitions'][event] += \
          [(action, self.current if target is None else target)]
        return self
```

N.B.: no high-level API is provided for 'observation'; one would access the dictionary directly.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An interpreter (a 'simulator') in Python

```
def run(fsm, input):
    # Determine initial state
    for id, [decl] in fsm.iteritems():
        if decl["initial"]:
            current = decl
            break
    # Consume input; produce output
    output = []
    while input:
        event = input.pop(0)
        if event not in current["transitions"]: raise FsmlInfeasibleEventException
        else:
            [(action, target)] = current["transitions"][event]
            if action is not None: output.append(action)
            if target not in fsm: raise FsmlResolutionException
            [current] = fsm[target]
    return output
```

N.B.: When compared to the Java-based interpreter, we access directly the presentation.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

‘Minimum’ DSL implementation

- ✓ Syntax (fluent API for internal DSL)
- ✓ (Dynamic) semantics (e.g., by means on an interpreter)
- \- Well-formedness / -typedness (aka static semantics)

N.B.: Just like the interpreter, we implement a ‘well-formedness checker’ as functionality on top of (the API for) the internal DSL representation. (We could use a constraint language such as OCL.)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Well-formedness of FSMs

distinctStateIds The state ids of the state declarations must be distinct.
singleInitialState An FSM must have exactly one initial state.
deterministicTransitions The events must be distinct per state.
resolvableTargetStates The target state of each transition must be declared.
reachableStates All states must be reachable from the initial state.

```
resolutionNotOk = \
  Fsm() \
    .addState("stateA") \
      .addTransition("eventI", "actionI", "stateB") \
      .addTransition("eventII", "actionII", "stateC") \
    .addState("stateB")
```

N.B.: This sample violates *resolvableTargetStates*.

N.B.: a violated *resolvableTargetStates* can (should) be detected even before running an FSM on a specific input.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Python-based well-formedness checker

```
def ok(fsm):
    for fun in [
            distinctStateIds,
            singleInitialState,
            deterministicTransitions,
            resolvableTargetStates,
            reachableStates ] : fun(fsm)
```

N.B.: Violations of *distinctStateIds* and *deterministicTransitions* can be detected during construction, but we may need explicit checks if we also accommodate 'serialization'.

```
def distinctStateIds(fsm):
    for state, decls in fsm.iteritems():
        if not len(decls) == 1: raise FsmlDistinctIdsException()

def singleInitialState(fsm):
    initials = [initial for initial, [decl] in fsm.iteritems() if decl["initial"]]
    if not len(initials) == 1: raise FsmlSingleInitialException()

def deterministicTransitions(fsm):
    for state, [decl] in fsm.iteritems():
        for event, transitions in decl["transitions"].iteritems():
            if not len(transitions) == 1: raise FsmlDeterminismException()

def resolvableTargetStates(fsm):  ...
```

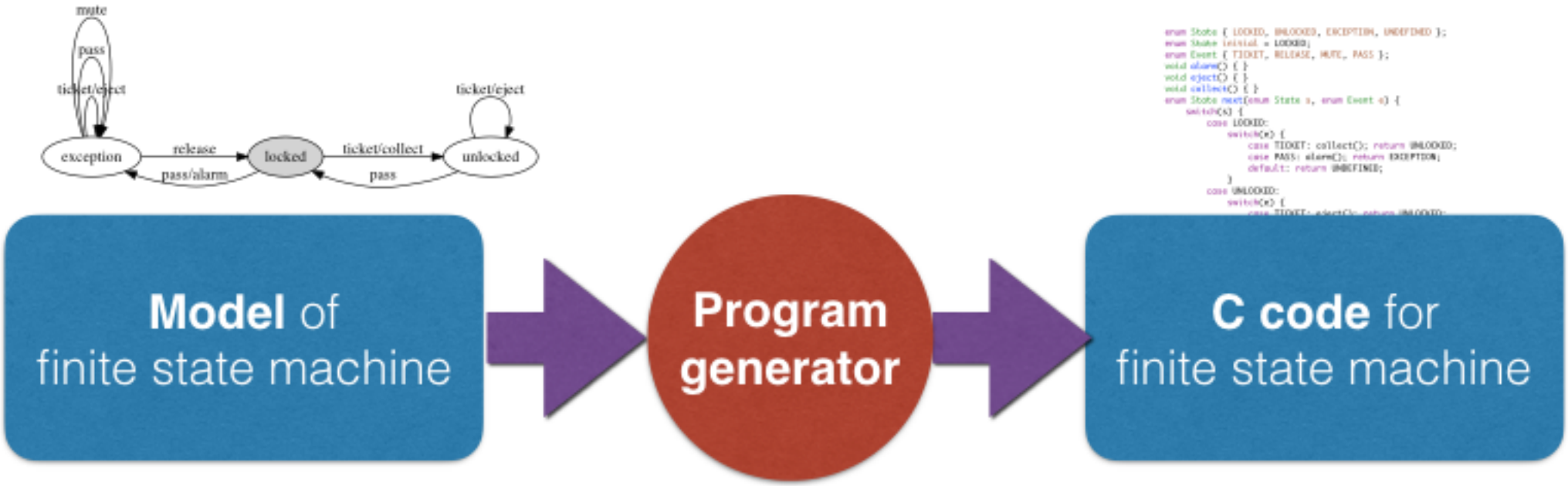
```
def singleInitialState(fsm):
    initials = [initial for initial, [decl] in fsm.iteritems() if decl["initial"]]
    if not len(initials) == 1: raise FsmlSingleInitialException()

def deterministicTransitions(fsm):
    for state, [decl] in fsm.iteritems():
        for event, transitions in decl["transitions"].iteritems():
            if not len(transitions) == 1: raise FsmlDeterminismException()

def resolvableTargetStates(fsm):
    for _, [decl] in fsm.iteritems():
        for _, transitions in decl["transitions"].iteritems():
            for (_, target) in transitions:
                if not target in fsm: raise FsmlResolutionException()

def reachableStates(fsm):
    for initial, [decl] in fsm.iteritems():
        if decl["initial"]:
            reachables = set([initial])
            chaseStates(initial, fsm, reachables)
    if not reachables == set(fsm.keys()): raise FsmlReachabilityException()

# Helper for recursive closure of reachable states
def chaseStates(source, fsm, states): ...
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Code generation for DSLs

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

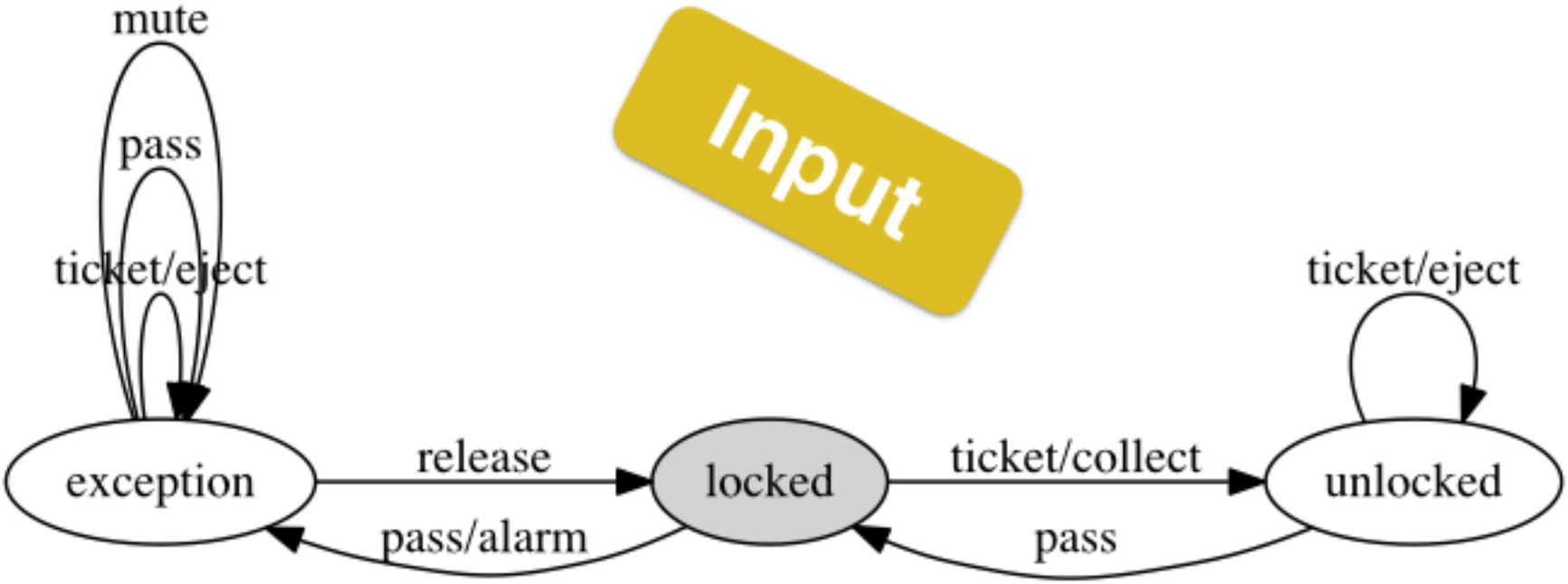
Problem statement



N.B.: The C code should run on some ‘target hardware’.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An FSM for a turnstile in a metro



- **States** (nodes): locked, unlocked, exception
- **Events**: ticket, pass, release, mute
- **Actions**: collect, eject, alarm
- **Transitions** (edges)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

```
enum State { LOCKED, UNLOCKED, EXCEPTION, UNDEFINED };
enum State initial = LOCKED;
enum Event { TICKET, RELEASE, MUTE, PASS };
void alarm() { }
void eject() { }
void collect() { }
enum State next(enum State s, enum Event e) {
    switch(s) {
        case LOCKED:
            switch(e) {
                case TICKET: collect(); return UNLOCKED;
                case PASS: alarm(); return EXCEPTION;
                default: return UNDEFINED;
            }
        case UNLOCKED:
            switch(e) {
                case TICKET: eject(); return UNLOCKED;
                case PASS: return LOCKED;
                default: return UNDEFINED;
            }
        case EXCEPTION:
            switch(e) {
                case TICKET: eject(); return EXCEPTION;
                case PASS: return EXCEPTION;
                case MUTE: return EXCEPTION;
                case RELEASE: return LOCKED;
                default: return UNDEFINED;
            }
        default: return UNDEFINED;
    }
}
```

C code for the turnstile FSM

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

How to design and implement the program generator?

- How to emit/produce/… the output—the actual C code?
  - We are going to use `template processing'!
- What data structure (model) to use for code generation?
  - Do we map FSML `objects' *directly* to C code?
    - No — the objects are not aligned with the wanted C code.
  - Do we build an `intermediate representation'?
    - Yes — we use a model close to the wanted C code.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

‘Hello, World’ of template processing

Source: https://github.com/antlr/stringtemplate4/blob/master/doc/introduction.md

Templates contain text and references to names:

```
Hello, <name>
```

Actual parameters can be passed to a rendering process

```
import org.stringtemplate.v4.*;
...
ST hello = new ST("Hello, <name>");
hello.add("name", "World");
System.out.println(hello.render());
```

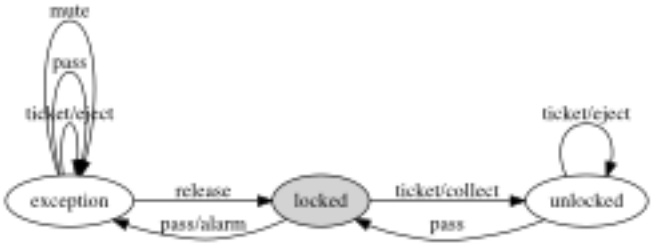
The rendering result is text.

```
Hello, World
```

N.B.: There exist many template processors. We use StringTemplate http://www.stringtemplate.org/ which is a mainstream option for the Java platform. StringTemplate can also be used with other languages.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Design and implement the code generator




```
enum State { LOCKED, UNLOCKED, EXCEPTION, UNDEFINED };
enum State initial = LOCKED;
enum Event { TICKET, RELEASE, MUTE, PASS };
void alarm() { }
void eject() { }
void collect() { }
enum State next(enum State s, enum Event e) {
    switch(s) {
        case LOCKED:
            switch(e) {
                case TICKET: collect(); return UNLOCKED;
                case PASS: alarm(); return EXCEPTION;
                default: return UNDEFINED;
            }
        case UNLOCKED:
            switch(e) {
```

Model of finite state machine

Program generator

C code for finite state machine

Define **templates** for the output

Map model to **intermediate representation**

Instantiation (rendering)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

// Main template for complete source code

```
main(states, initial, events, actions, tgroups) ::= << ... >>
```

// Template for functions that implement actions

```
action(a) ::= "..."
```

// switch-case for transitions grouped by source state

```
tgroup(g) ::= <<...>>
```

// case for a specific transition

```
transition(t) ::= <%...%>
```

Outline of the **templates** for generating C code

N.B.: Because the templates have names, they can refer to each other. This is called a *template group* in StringTemplate.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

```
main(states, initial, events, actions, tgroups) ::= <<
enum State { <states; format="upper", separator=", "> };
enum State initial = <initial; format="upper">;
enum Event { <events; format="upper", separator=", "> };
<actions:action(); format="lower", separator="\n">
enum State next(enum State s, enum Event e) {
    switch(s) {
<tgroups:tgroup(); separator="\n">
        default: return UNDEFINED;
    }
}>>

action(a) ::= "void <a>() { }"

tgroup(g) ::= <<
        case <g.stateid; format="upper">:
            switch(e) {
                <g.ts:transition(); separator="\n">
                default: return UNDEFINED;
            }>>

transition(t) ::= <%
case <t.event; format="upper">:
<if(t.action)><t.action; format="lower">(); <endif>
return <t.target; format="upper">;%>
```

The templates for generating C code

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A model for finite state machines

```
public class Fsm {
        public List<State> states = new LinkedList<>();
        public List<Transition> transitions = new LinkedList<>();
}

public class State {
        public String stateid;
        public boolean initial;
}

public class Transition {
        public String source;
        public String event;
        public String action;
        public String target;
}
```

N.B.: In reality, we use private fields, public getters, and possibly functional constructors instead of public fields. In fact, we may also assume a fluent API.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

```
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;
import org.stringtemplate.v4.StringRenderer;
import java.io.File;
import java.util.*;

public class FsmlCGenerator {
        private static class TGroup {
                public String stateid;
                public List<Transition> ts;
        }
        public static String generateC(Fsm fsm) {
                // Build list of states with extra "UNDEFINED"
                List<String> states = ...
                // Build set of events
                Set<String> events = ...
                // Build set of actions
                Set<String> actions = ...
                // Group transitions by state
                List<TGroup> tgroups = ...
                // Load template group and retrieve top-level template
                STGroup group = new STGroupFile(... + "Fsm.stg");
                group.registerRenderer(String.class, new StringRenderer());
                ST main = group.getInstanceOf("main");
                // Set template parameters and render
                main.add("states", states);
                main.add("initial", fsm.getInitial());
                main.add("events", events);
                main.add("actions", actions);
                main.add("tgroups", tgroups);
                return main.render();
        }
}
```

Intermediate representation

Map model to intermediate representation

Instantiation (rendering)

Outline of the Java code for **program generation**

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

```
// Build list of states with extra "UNDEFINED"
List<String> states = new LinkedList<>();
for (State s : fsm.getStates()) states.add(s.getStateid());
states.add("UNDEFINED");
// Build set of events
Set<String> events = new HashSet<>();
for (Transition t : fsm.getTransitions()) events.add(t.getEvent());
// Build set of actions
Set<String> actions = new HashSet<>();
for (Transition t : fsm.getTransitions())
    if (t.getAction()!=null) actions.add(t.getAction());
// Group transitions by state
List<TGroup> tgroups = new LinkedList<>();
for (State s : fsm.getStates()) {
    TGroup tg = new TGroup();
    tg.stateid = s.getStateid();
    tg.ts = new LinkedList<>();
    for (Transition t : fsm.getTransitions())
        if (tg.stateid==t.getSource()) tg.ts.add(t);
    tgroups.add(tg);
}
```

Map model to intermediate representation

N.B.: We aim at a strict separation of model and view and thus, we do not try to ‘compute’ anything in the template.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

StringTemplate's metanotation

Notation	Meaning
tmpl(*p*) ::= "…"	*Define a template tmpl with parameter p*
tmpl(*p*) ::= <<…>>	Multiline template with indentation and linebreaks
tmpl(*p*) ::= <%…%>	Ignore indentation and linebreaks
<*p*>	Refer to parameter *p*
<*p*; format="upper">	Format in uppercase
<*p*; separator=", ">	Separate elements of collection by comma
<*p*:*tmpl*()>	Apply *tmpl* on (each element of) *p*
<*p*.*x*>	Refer to property *x* of *p*
<if(*p*.*x*)>…<endif>	Include text only if *p*.*x* is true or not NULL

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Template processing issues

- **Escaping** template metanotation
- Spacing, **indentation**, and linebreaks
- Underlying use of **reflection** (typically)
- **Formatting** (e.g., upper- versus lowercase)
- **Mapping** the model to the template parameters
- **Idiosyncrasies** of particular template metanotation

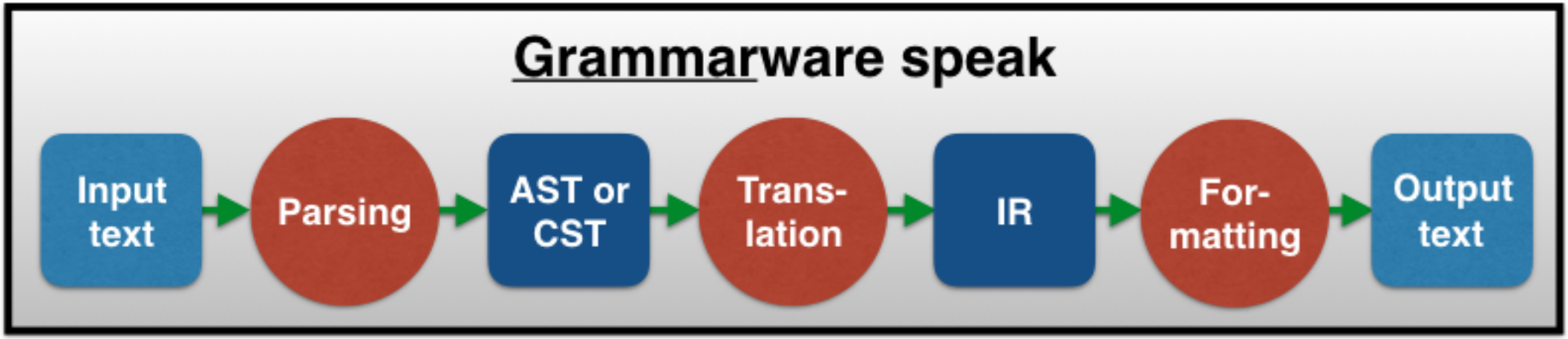
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A Jinja2 template for use with Python

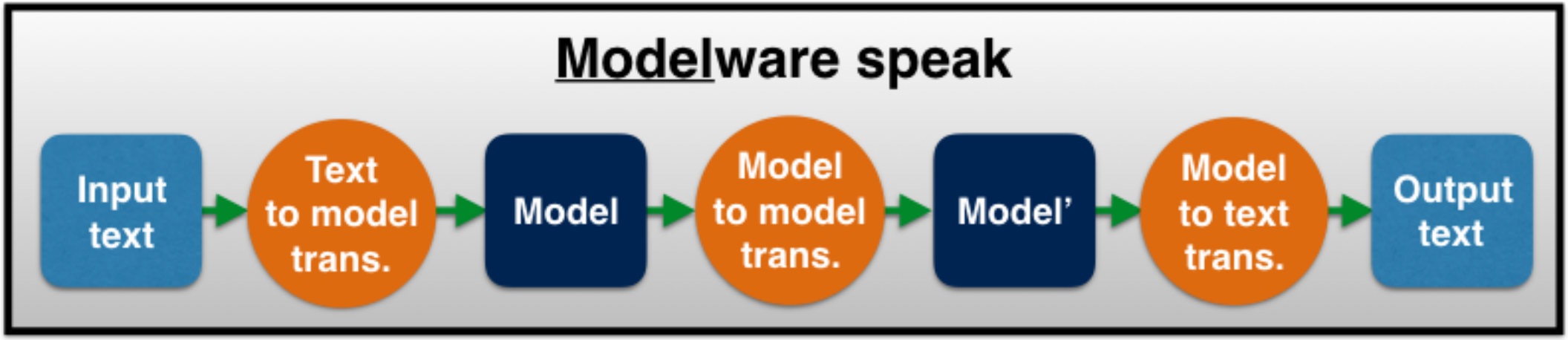
```
enum State { {{states|join(', ')|upper()}} };
enum State initial = {{initial|upper}};
enum Event { {{events|join(', ')|upper()}} };

{% for a in actions %}void {{a}}() { }
{% endfor %}
enum State next(enum State s, enum Event e) {
  switch(s) { {% for (s, ts) in transitions %}
  case {{s|upper()}}:
    switch(e) { {% for (e, a, t) in ts %}
    case {{e|upper()}}: {% if a %}{{a}}(); {%endif%}return {{t|upper()}};{% endfor
        %}
    default: return UNDEFINED;
    }{% endfor %}
  default: return UNDEFINED;
  }
}
```

N.B.: This approach slightly differs from the one shown before.
In particular, we use a for-loop.
Also, we use a single template as opposed to a group.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.



versus

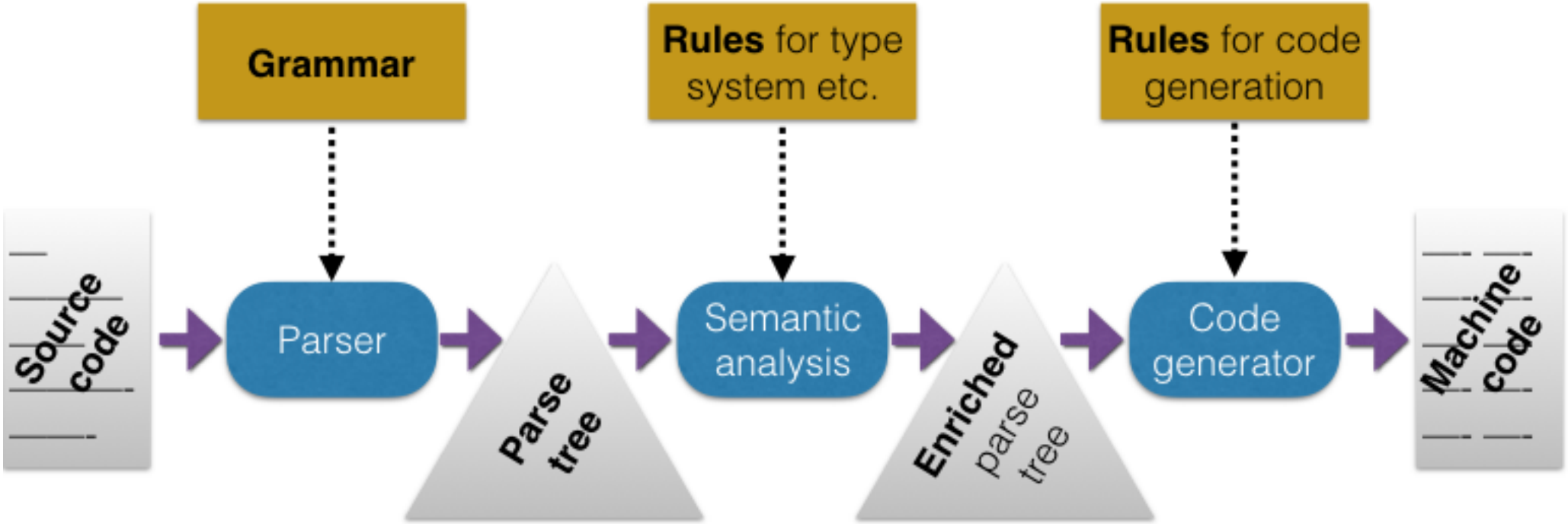


N.B.:

- IR = Intermediate Representation (an AST closer to output).
- We massage FSML 'models' to appeal to code generation patterns.
- We could also translate FSML 'models' directly into C 'models'.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Comparison with data flow in a compiler

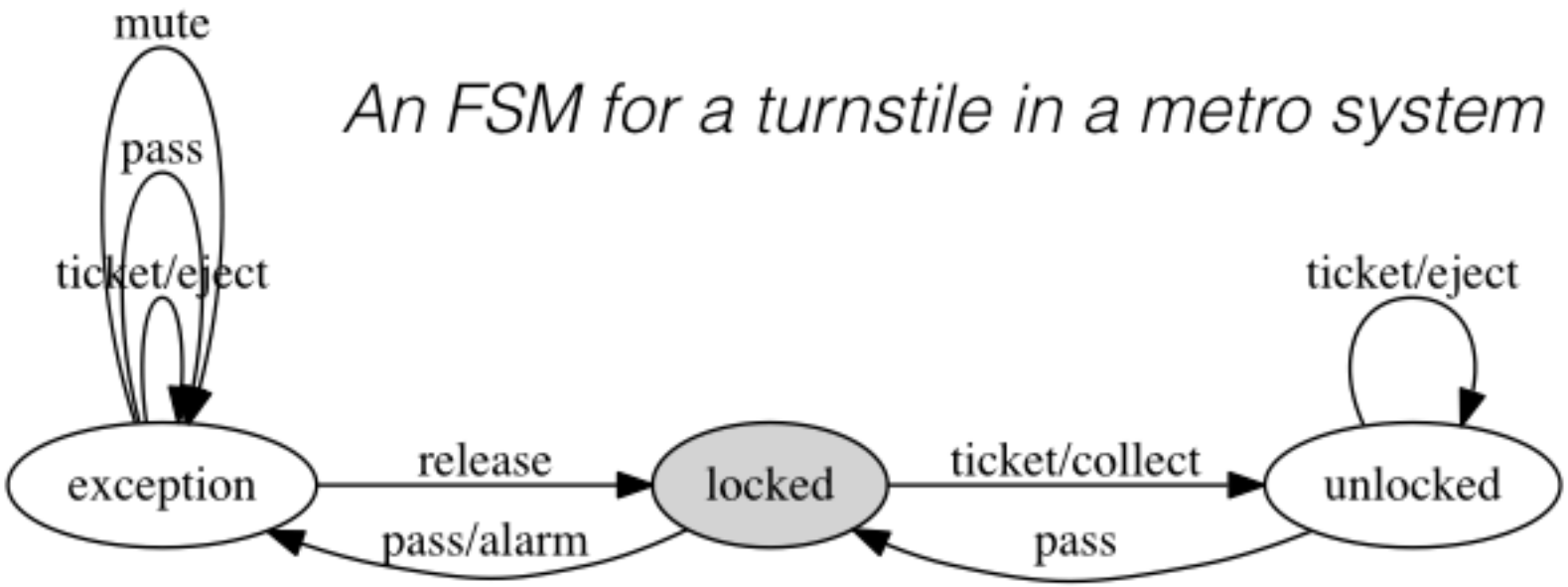


© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

External DSL style

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Let's have a textual syntax for FSML: the finite state machine (FSM) language



An FSM for a turnstile in a metro system

- **States** (nodes): locked, unlocked, exception
- **Events**: ticket, pass, release, mute
- **Actions**: collect, eject, alarm
- **Transitions** (edges)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Proposal for a textual syntax for FSML

```
initial state locked {
  ticket/collect -> unlocked;
  pass/alarm -> exception;
}
state unlocked {
  ticket/eject;
  pass -> locked;
}
state exception {
  ticket/eject;
  pass;
  mute;
  release -> locked;
}
```

N.B.: DSL design
(concrete or abstract syntax design)
commences in a sample-driven manner.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

DSL implementation in different ‘styles’

Our shifted focus

- **External DSL**:

  Designated parser, checker, interpreter, compiler

- **Internal DSL**:

  Implementation as library using host language features

N.B.: This is a gross oversimplification.
There are options or hybrids using extensible languages, extensible compilers, metaprogramming systems, and language workbenches.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

We are going to do here …

External DSL style

with

and

N.B.: We are committing to a particular parser generator (ANTLR). We could also be using hand-written parsers, parser combinators, and model-to-text technologies. ANTLR, by itself, also serves other 'target' languages, e.g., Python.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Grammar-based concrete syntax definition

```
fsm : state+ EOF ;
state : 'initial'? 'state' stateid '{' transition* '}' ;
transition : event ('/' action)? ('->' target=stateid)? ';' ;
stateid : NAME ;
event : NAME ;
action : NAME ;
NAME : ('a'..'z'|'A'..'Z')+ ;
```

N.B.: Action and target state are optional.

N.B.: This is essentially Extended Backus Naur Form.
‘?’ for options, ‘*’/‘+’ for lists, etc.
Well, we use the specific grammar notation of ANTLR.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

ANTLR+Java-based syntax checker

```
grammar Fsml;
@header {package org.softlang.fsml;}
fsm : state+ EOF ;
state : 'initial'? 'state' stateid '{' transition* '}' ;
transition : event ('/' action)? ('->' target=stateid)? ';' ;
stateid : NAME ;
event : NAME ;
action : NAME ;
NAME : ('a'..'z'|'A'..'Z')+ ;
WS : [ \t\n\r]+ -> skip ;
```

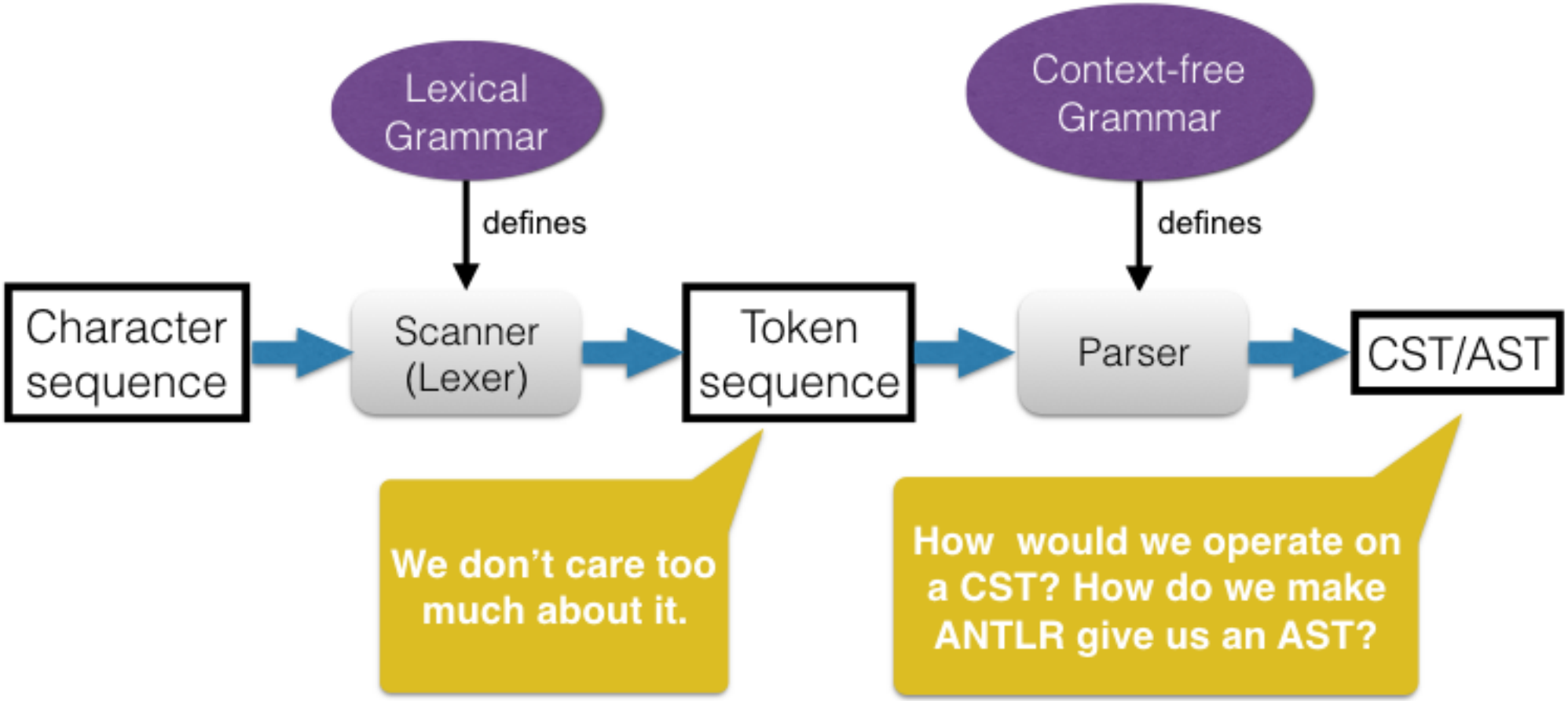
Package for generated code

Skip whitespace!

N.B.: A grammar is almost an effective definition of a syntax checker. We need ‘pragmas’ and driver code (see next slide).

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Data flow in (scannerfull) parsing



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Sample FSM

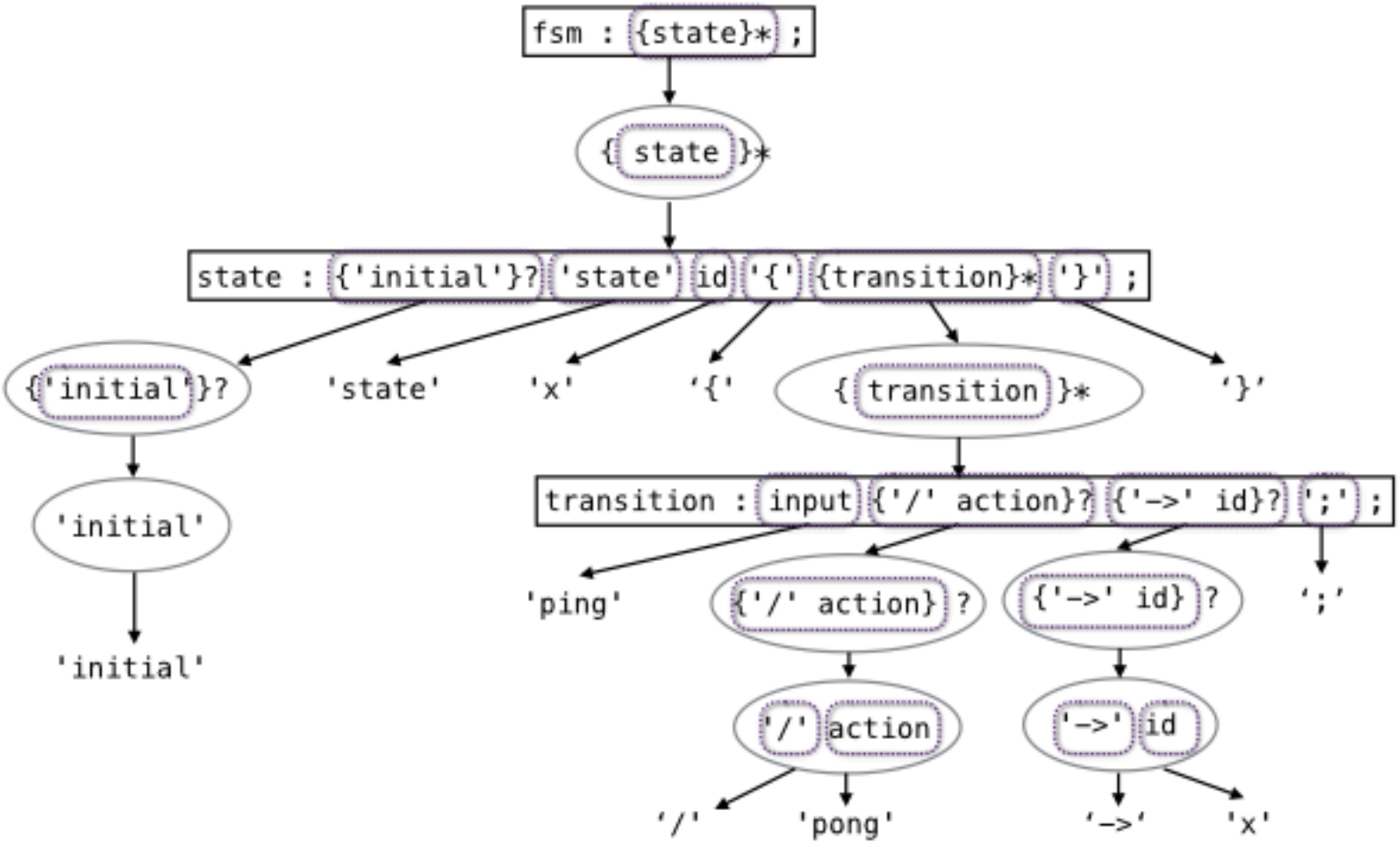
```
initial state x {
    ping | pong -> x
}
```

Token stream

terminal(“initial”),
terminal(“state”),
NAME(”x”),
terminal(“{“),
NAME(“ping”),
terminal(“/“),
NAME(“pong”),
terminal(“->”),
NAME(“x”),
terminal(“}”)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

CST for sample FSM

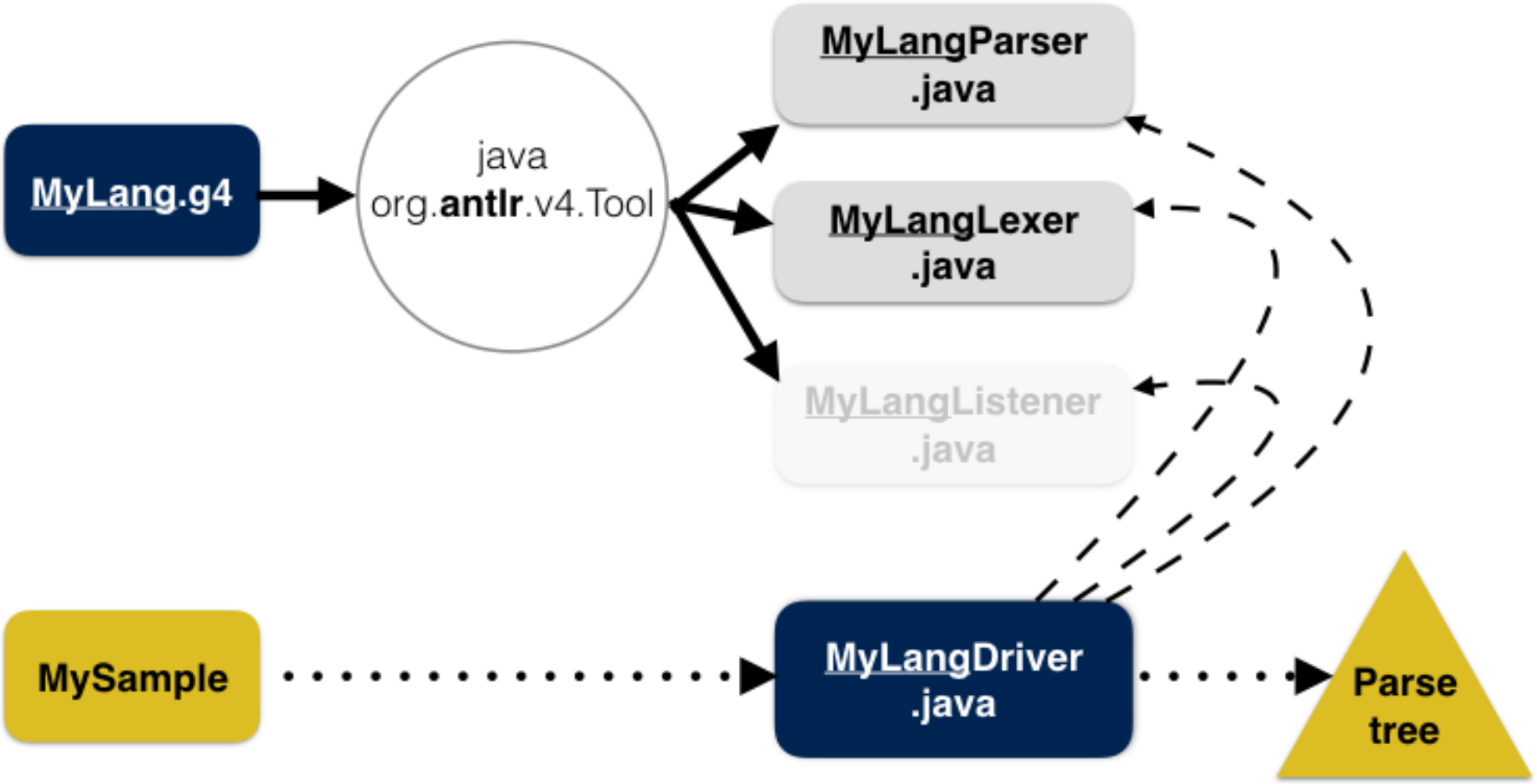


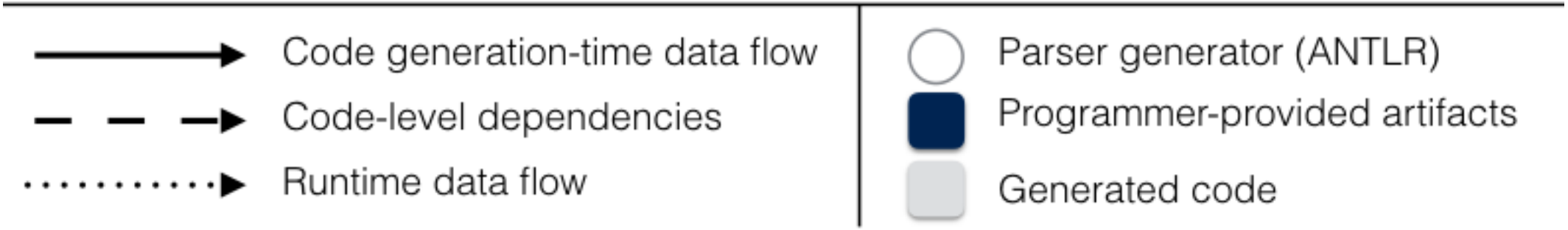
N.B.:

- The leaf infos represent the input string.
- The inner nodes correspond to rule applications.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Parser generation with ANTLR





© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Java code driving the syntax checker

based on generated classes *Parser and *Lexer

```
public class FsmlSyntaxChecker {
  public static void main(String[] args) throws IOException {
    FsmlParser parser =
      new FsmlParser(
        new CommonTokenStream(
          new FsmlLexer(
            new ANTLRFileStream(args[0]))));
    parser.fsm();
    System.exit(parser.getNumberOfSyntaxErrors()-Integer.parseInt(args[1]));
  }
}
```

- Token stream to parser
- Lexer to token stream
- Stream to lexer
- Filename to stream

We assume a command-line interface for running positive and negative test cases.

N.B.: This is boilerplate code.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

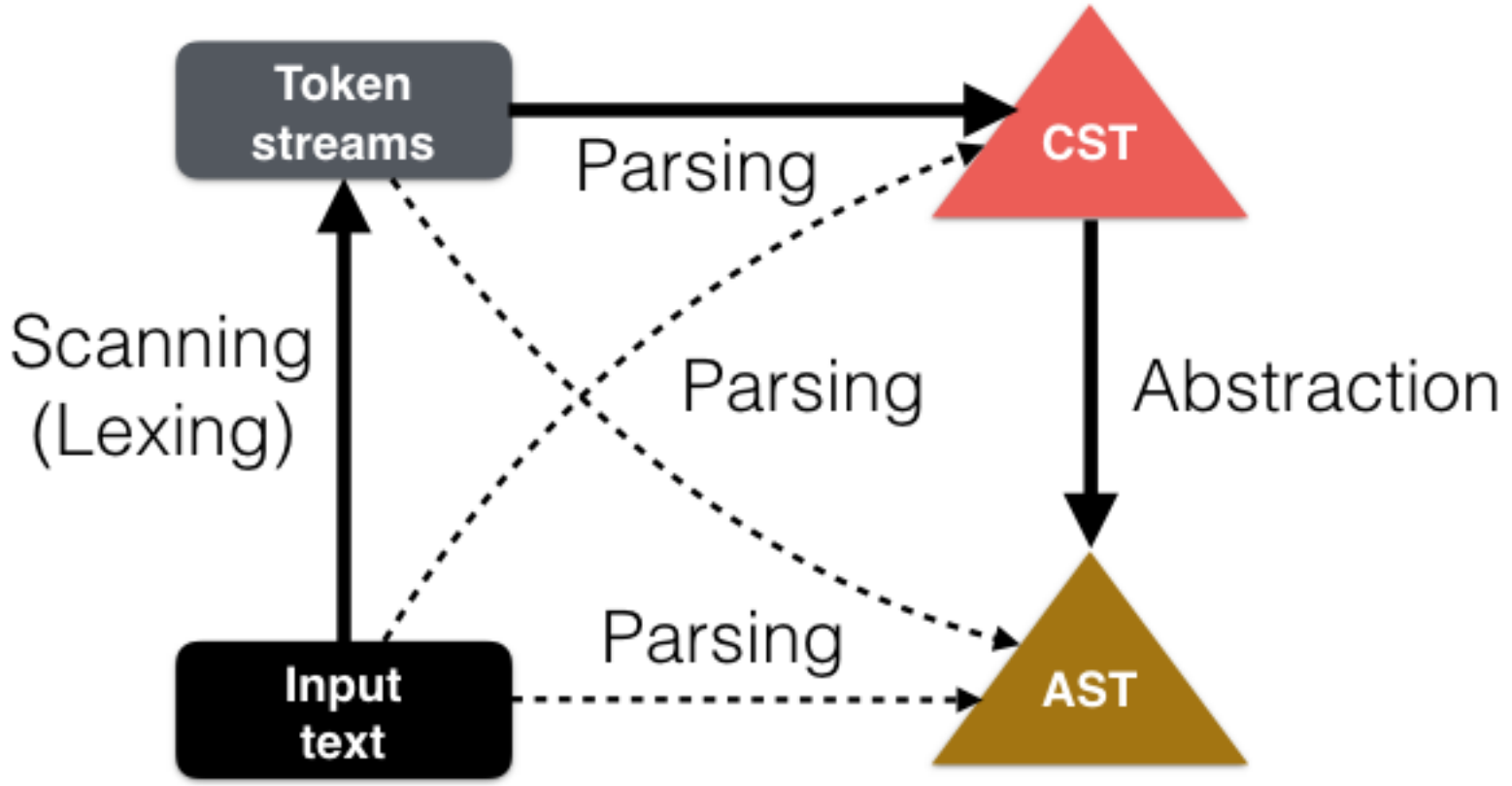
Makefile
for building and testing the syntax checker

```
cp = -cp .:../../../lib/Java/antlr-4.5.3-complete.jar
antlr = java ${cp} org.antlr.v4.Tool -o org/softlang/fsml
fsmlSyntaxChecker = java ${cp} org.softlang.fsml.FsmlSyntaxChecker

all:
  make generate
  make compile
  make test

generate:
  ${antlr} Fsml.g4

compile:
  javac ${cp} org/softlang/fsml/*.java

test:
  ${fsmlSyntaxChecker} ../sample.fsml 0
  ${fsmlSyntaxChecker} ../tests/syntaxError.fsml 1
```

N.B.: Automation (build management and testing) is crucial in DSL implementation to deal with experimentation and evolution.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

From checking to parsing

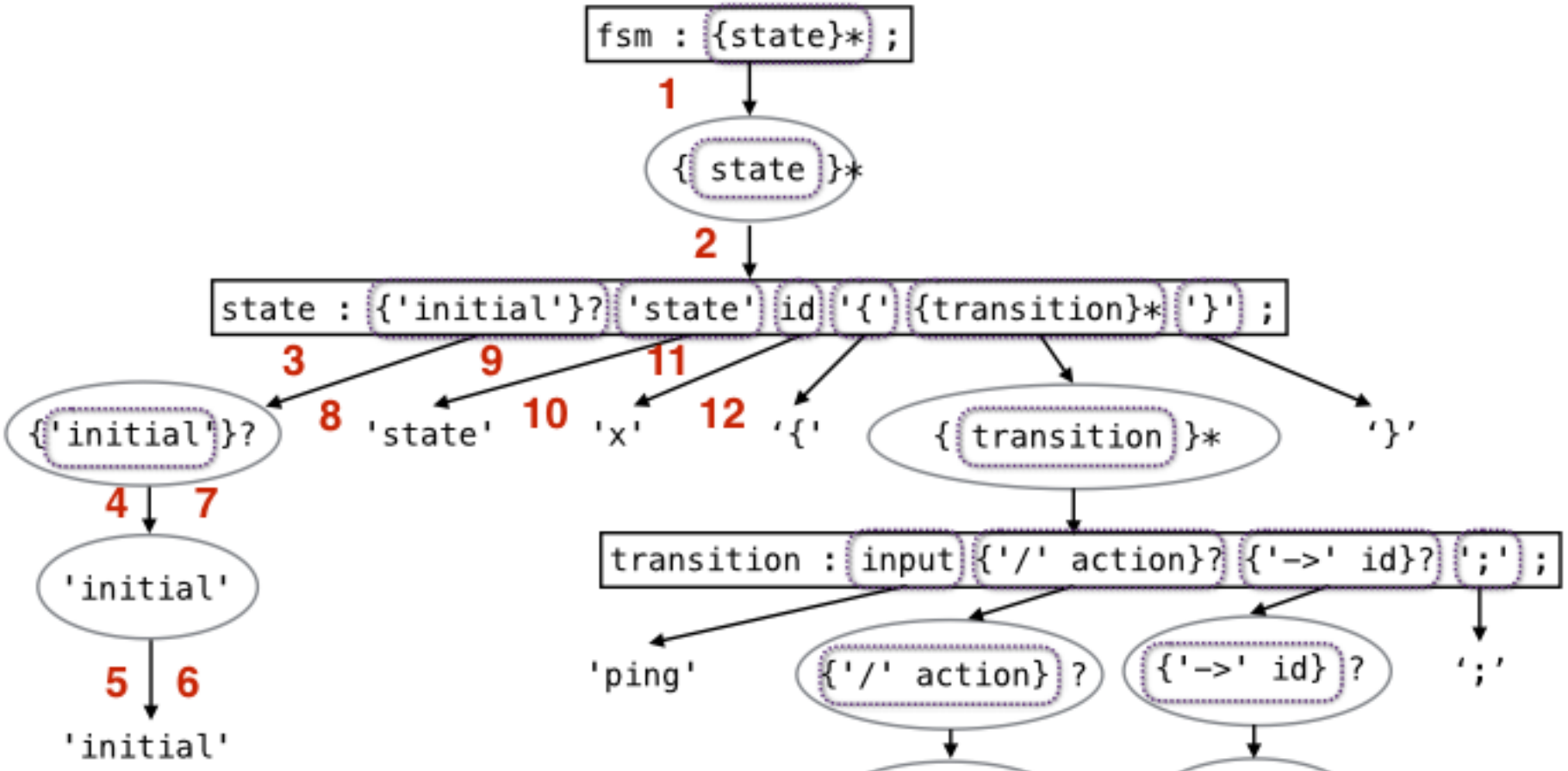


N.B.:

- AST/CST = abstract/concrete syntax tree
- CSTs are served by a parsing technology like ANTLR.
- ASTs are modeled by language-specific object models.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Abstraction based on walking the CST, in fact, based on ‘listening’ to a walk



N.B.: These are the first few steps of a depth-first walk.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An ANTLR listener for abstraction

```
public class FsmlToObjects extends FsmlBaseListener {
  private Fsm fsm;
  private State current;
  public Fsm getFsm() { return fsm; }
  @Override public void enterFsm(FsmlParser.FsmContext ctx) {
    fsm = new Fsm();
  }
  @Override public void enterState(FsmlParser.StateContext ctx) {
    current = new State();
    current.setStateid(ctx.stateid().getText());
    fsm.getStates().add(current);
  }
  @Override public void enterTransition(FsmlParser.TransitionContext ctx) {
    Transition t = new Transition();
    ...
```

N.B.:

- There are 'events' for entering (and leaving) nonterminals.
- The listener applies to the grammar for syntax checking.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

```
public class FsmlToObjects extends FsmlBaseListener {
  private Fsm fsm;
  private State current;
  public Fsm getFsm() { return fsm; }
  @Override public void enterFsm(FsmlParser.FsmContext ctx) {
    fsm = new Fsm();
  }
  @Override public void enterState(FsmlParser.StateContext ctx) {
    current = new State();
    current.setStateid(ctx.stateid().getText());
    fsm.getStates().add(current);
  }
  @Override public void enterTransition(FsmlParser.TransitionContext ctx) {
    Transition t = new Transition();
    fsm.getTransitions().add(t);
    t.setSource(current.getStateid());
    t.setEvent(ctx.event().getText());
    if (ctx.action() != null) t.setAction(ctx.action().getText());
    t.setTarget(ctx.target != null ? ctx.target.getText() : current.getStateid());
  }
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Java code driving the parser

```
public Fsm textToObjects(String filename) throws IOException {
  FsmlParser parser = new FsmlParser(
      new CommonTokenStream(
          new FsmlLexer(
              new ANTLRFileStream(filename))));
  ParseTree tree = parser.fsm();
  assertEquals(0, parser.getNumberOfSyntaxErrors());
  FsmlToObjects listener = new FsmlToObjects();
  ParseTreeWalker walker = new ParseTreeWalker();
  walker.walk(listener, tree);
  return listener.getFsm();
}
```

N.B.: This is essentially the same code as for the earlier checker—except that we now walk the parse tree.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

‘Minimum’ DSL implementation

- Syntax:
  - Object model for abstract syntax
  - **Parser based on grammar for concrete (textual) syntax**
- (Dynamic) semantics:
  - Interpreter operating on abstract syntax (object model)
- Well-formedness/typedness (aka static semantics):
  - Checker operating on abstract syntax (object model)

N.B.: Everything not in bold face can be implemented in the same way as in a DSL implementation in internal style. (Clearly, we only consider here a particular approach to DSL implementation.)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Metaprogramming

(A comparison of approaches)

Acknowledgement: This is joint work with Simon Schauß, Johannes Härtel, Kevin Klein, Wojciech Kwasnik (all SoftLang), and Thorsten Berger (Chalmers).

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

(Comparison of) metaprogramming approaches

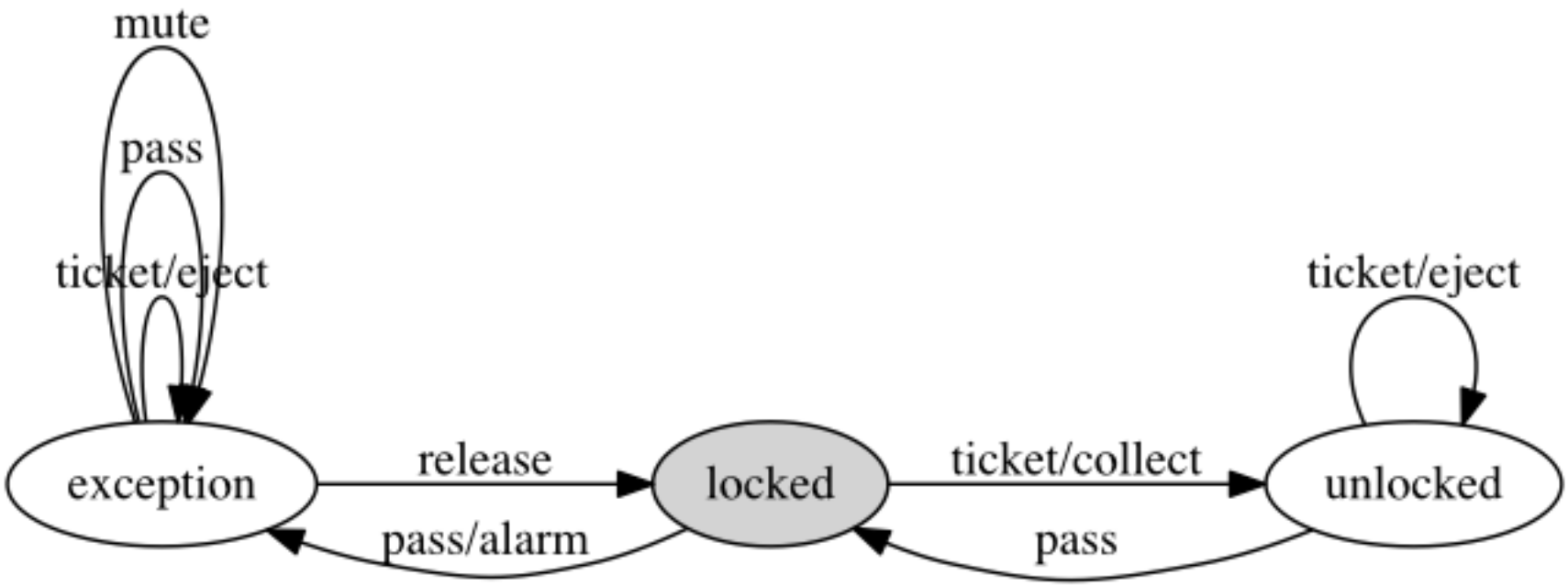
- A **metaprogram** consumes or produces (or analyzes or manipulates) programs.
- A **DSL implementation** (both in internal and external style) is a metaprogram.
- A **metaprogramming approach** is an approach to developing metaprograms, subject to designated language, library, and other tool support.
- **MetaLib** is a project for the comparison for metaprogramming approaches. In fact, it is a **chrestomathy**.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

The Finite State Machine Language (FSML)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Visual FSML notation



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Textual FSML notation

```
initial state locked {
  ticket/collect -> unlocked;
  pass/alarm -> exception;
}
state unlocked {
  ticket/eject;
  pass -> locked;
}
state exception {
  ticket/eject;
  pass;
  mute;
  release -> locked;
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Textual FSML notation

```
initial state locked {
  ticket/collect -> unlocked;
  pass/alarm -> exception;
}
state unlocked {
  ticket/eject;
  pass -> locked;
}
state exception {
  ticket/eject;
  pass;
  mute;
  release -> locked;
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Grammar of textual notation

```
fsm : {state}* ;
state : {'initial'}? 'state' stateid '{' {transition}* '}' ;
transition : event {'/' action}? {'->' stateid}? ';' ;
stateid : name ;
event : name ;
action : name ;
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Signature of abstract syntax

```
type fsm = state* ;
type state = initial × stateid × transition* ;
type initial = boolean ;
type transition = event × action? × stateid ;
type stateid = string ;
type event = string ;
type action = string ;
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Metamodel of abstract syntax

```
class fsm { part states : state* ; }
class state {
  value initial : boolean ;
  value stateid : string ;
  part transitions : transition* ;
}
class transition {
  value event : string ;
  value action : string? ;
  reference target : state ;
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Small-step operational semantics

$$\langle \ldots, \langle b, x, \langle \ldots, \langle e, \langle a \rangle, x' \rangle, \ldots \rangle \rangle, \ldots \rangle \vdash \langle x, e \rangle \rightarrow \langle x', \langle a \rangle \rangle \quad [\text{action}]$$

$$\langle \ldots, \langle b, x, \langle \ldots, \langle e, \langle \rangle, x' \rangle, \ldots \rangle \rangle, \ldots \rangle \vdash \langle x, e \rangle \rightarrow \langle x', \langle \rangle \rangle \quad [\text{no-action}]$$

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Well-formedness
(a violation thereof)

```
initial state stateA { eventI/actionI -> stateB; }
state stateB { }
state stateC { }
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

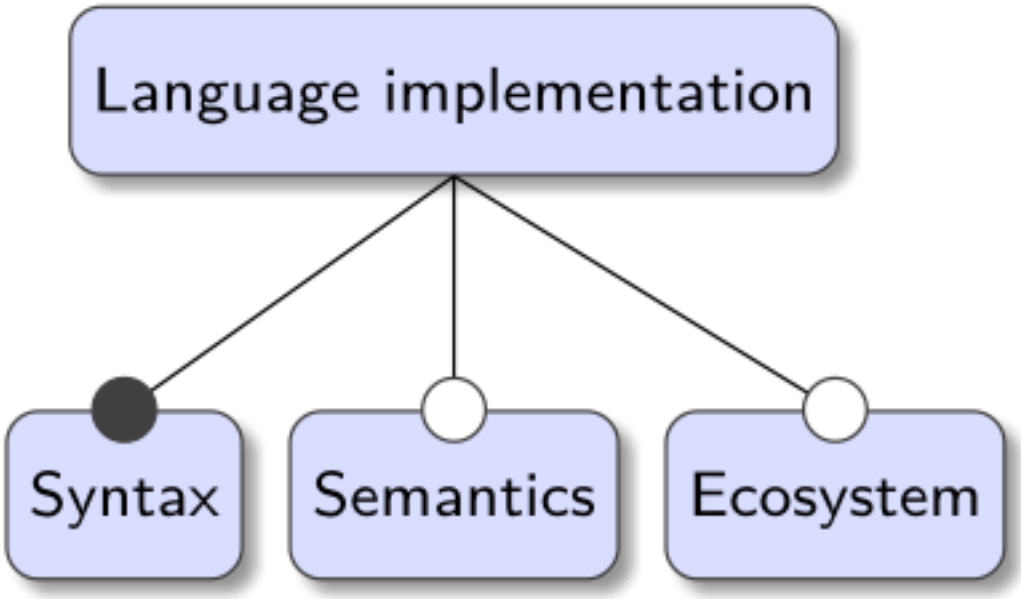
Generated C code

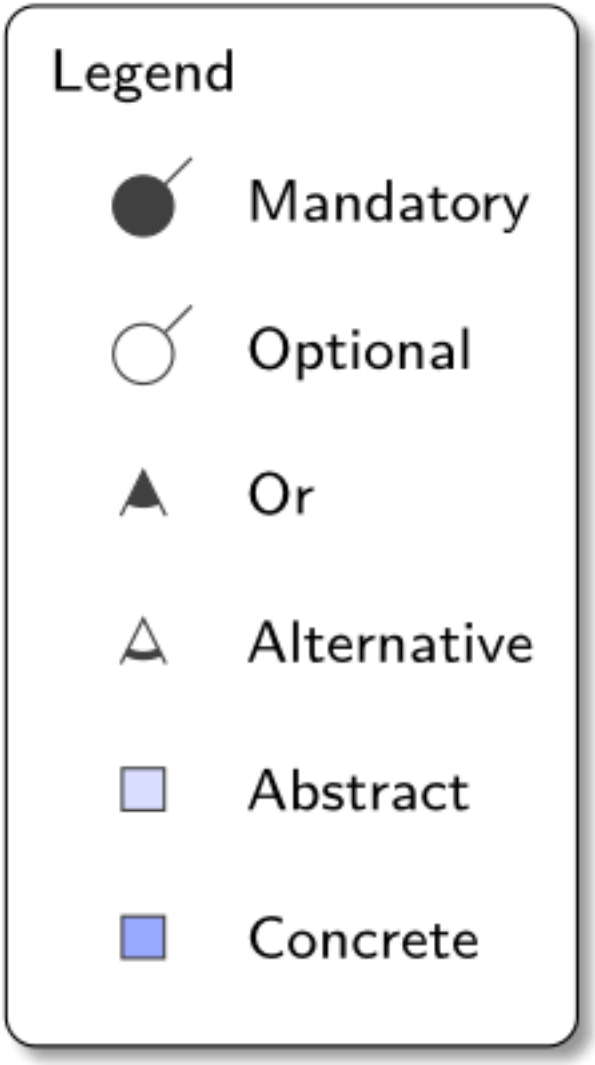
```
enum State {LOCKED,UNLOCKED,EXCEPTION,UNDEFINED};
enum State initial = LOCKED;
enum Event {TICKET,RELEASE,MUTE,PASS};
void alarm() { }
void eject() { }
void collect() { }
enum State next(enum State s, enum Event e) {
    switch(s) {
        case LOCKED:
            switch(e) {
                case TICKET: collect(); return UNLOCKED;
                case PASS: alarm(); return EXCEPTION;
                default: return UNDEFINED;
            }
        case UNLOCKED: ...
        case EXCEPTION: ...
        default: return UNDEFINED;
    }
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

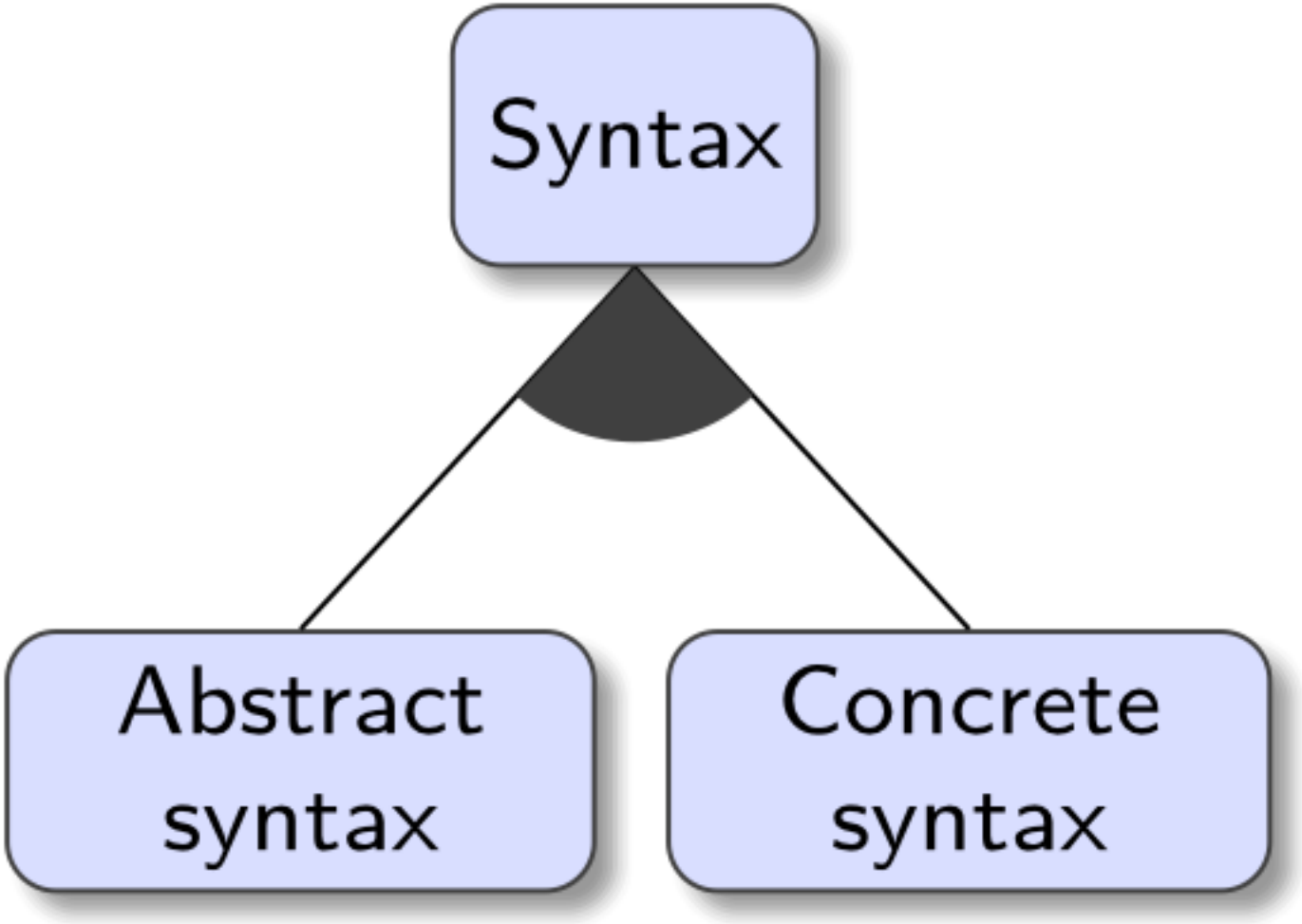
Feature model for the implementation of a DSL

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

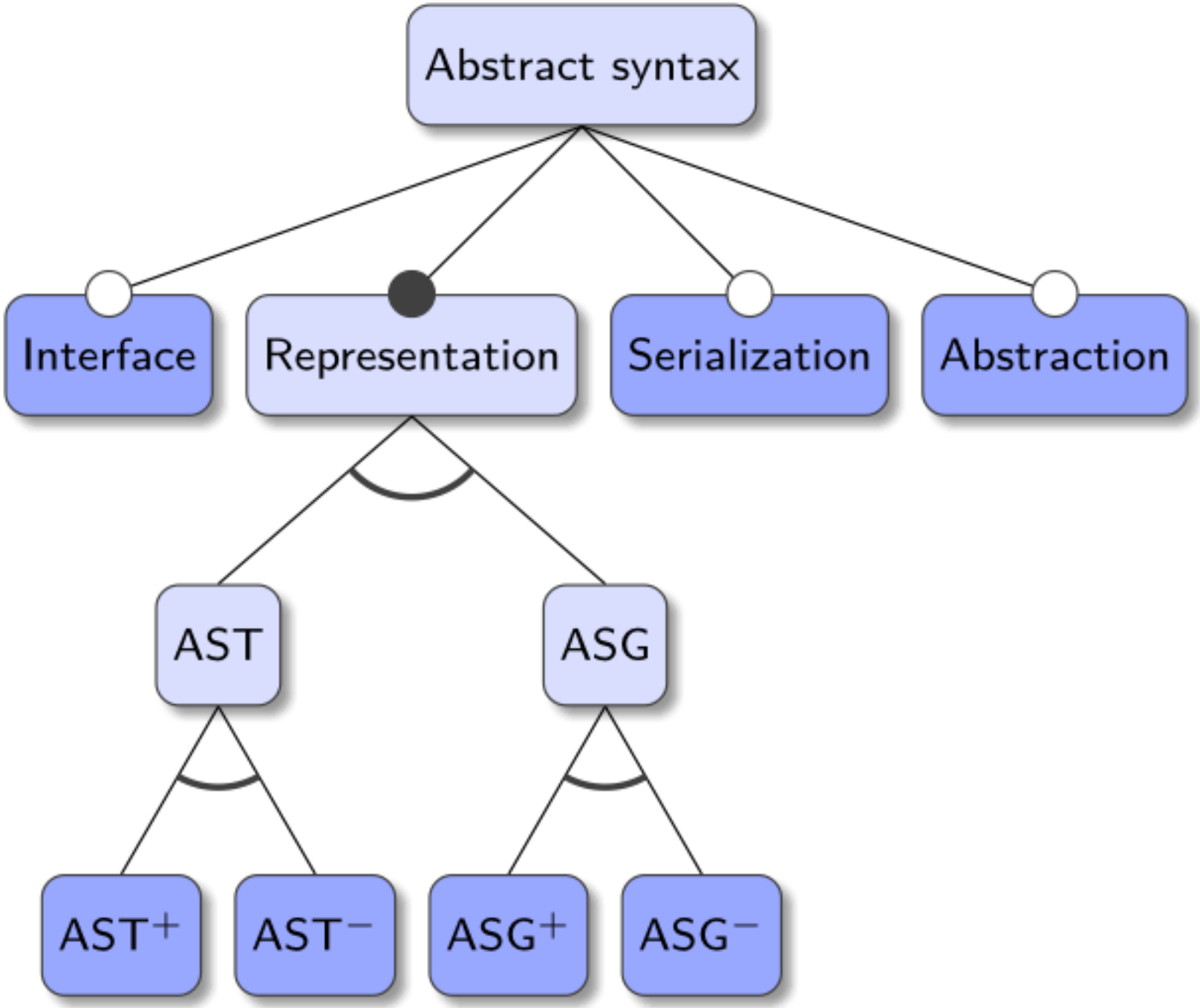




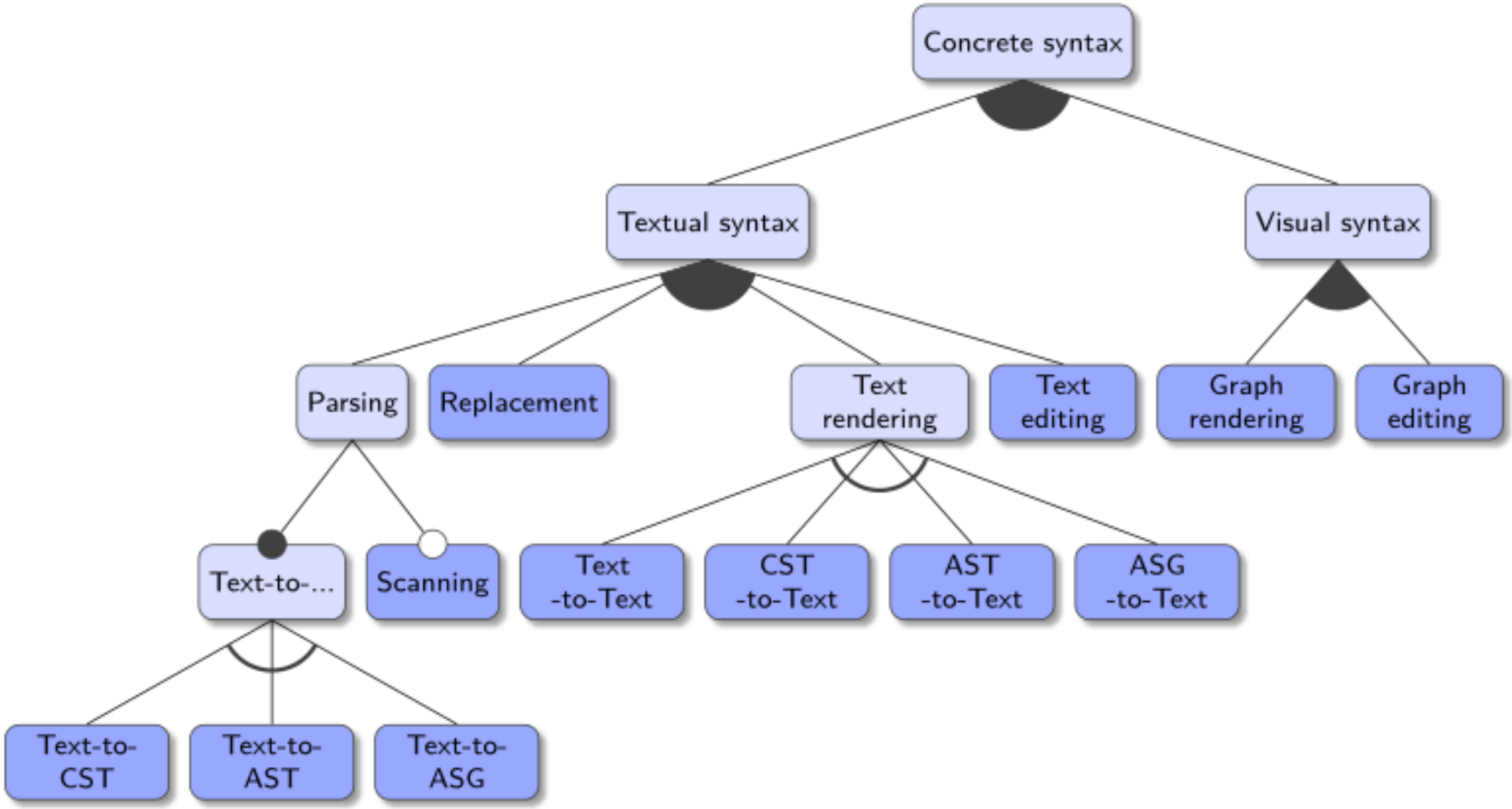
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

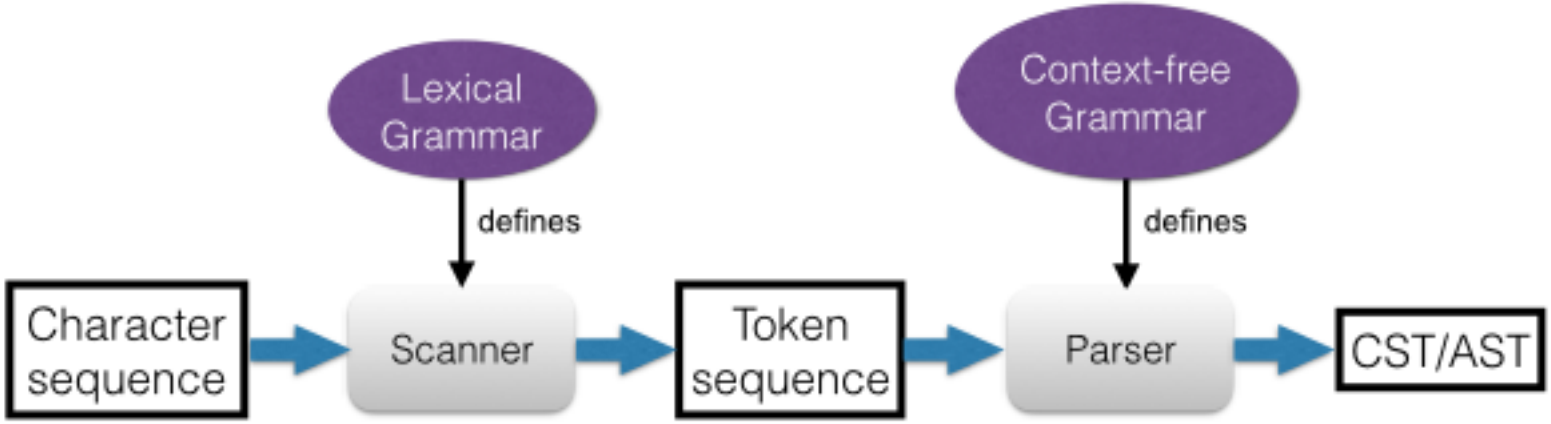


© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.



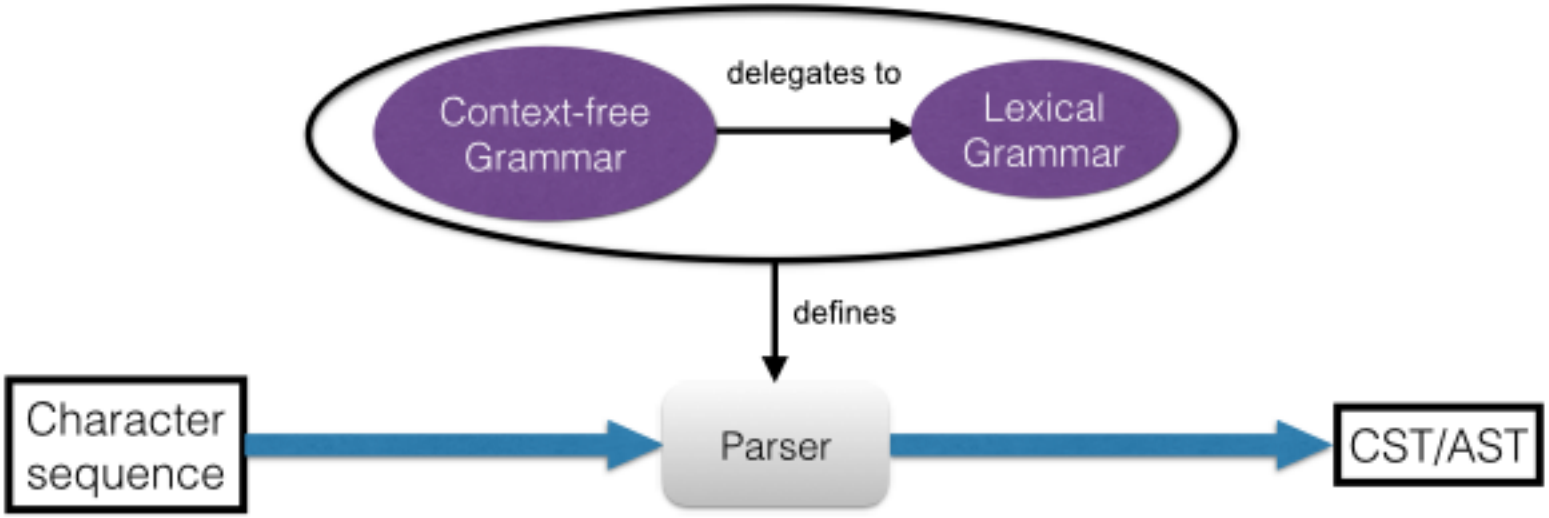
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Scannerfull parsing

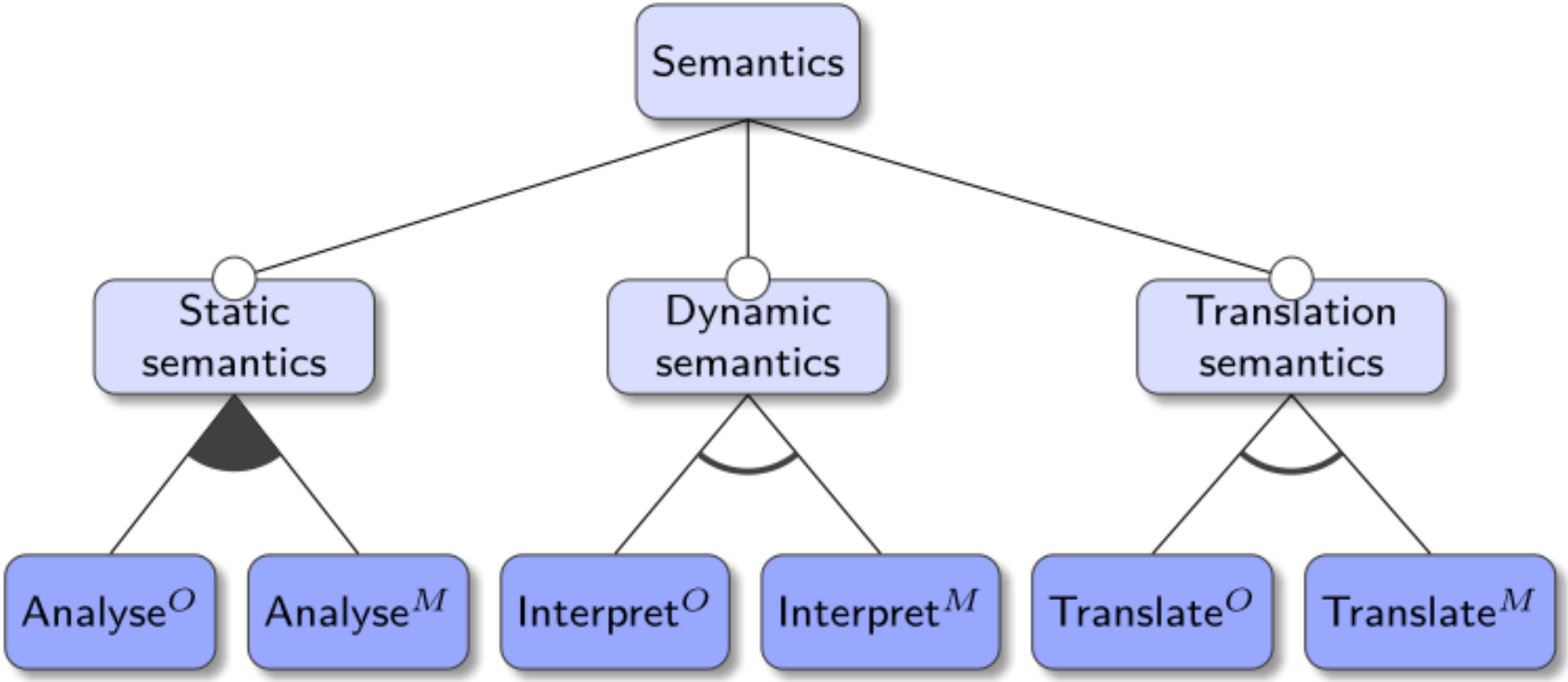


© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

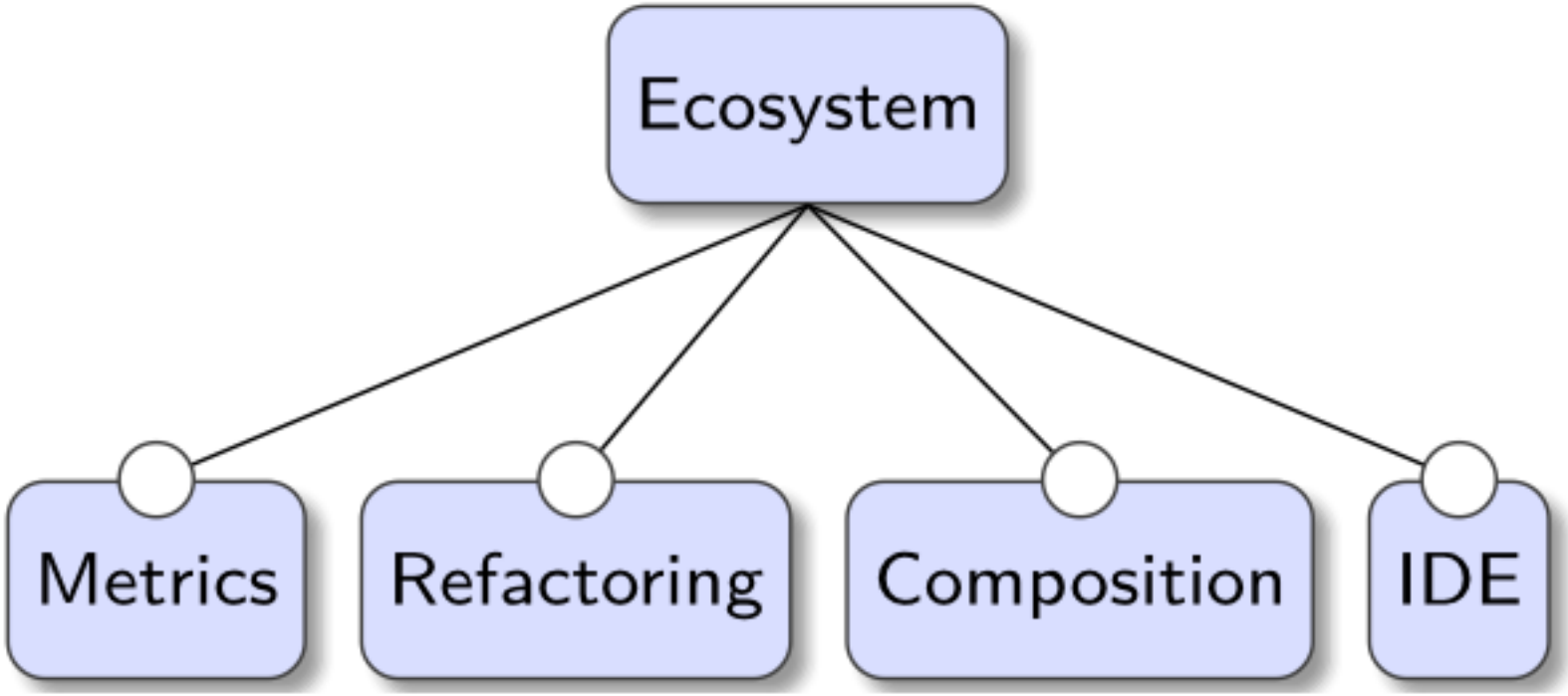
Scannerless parsing



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

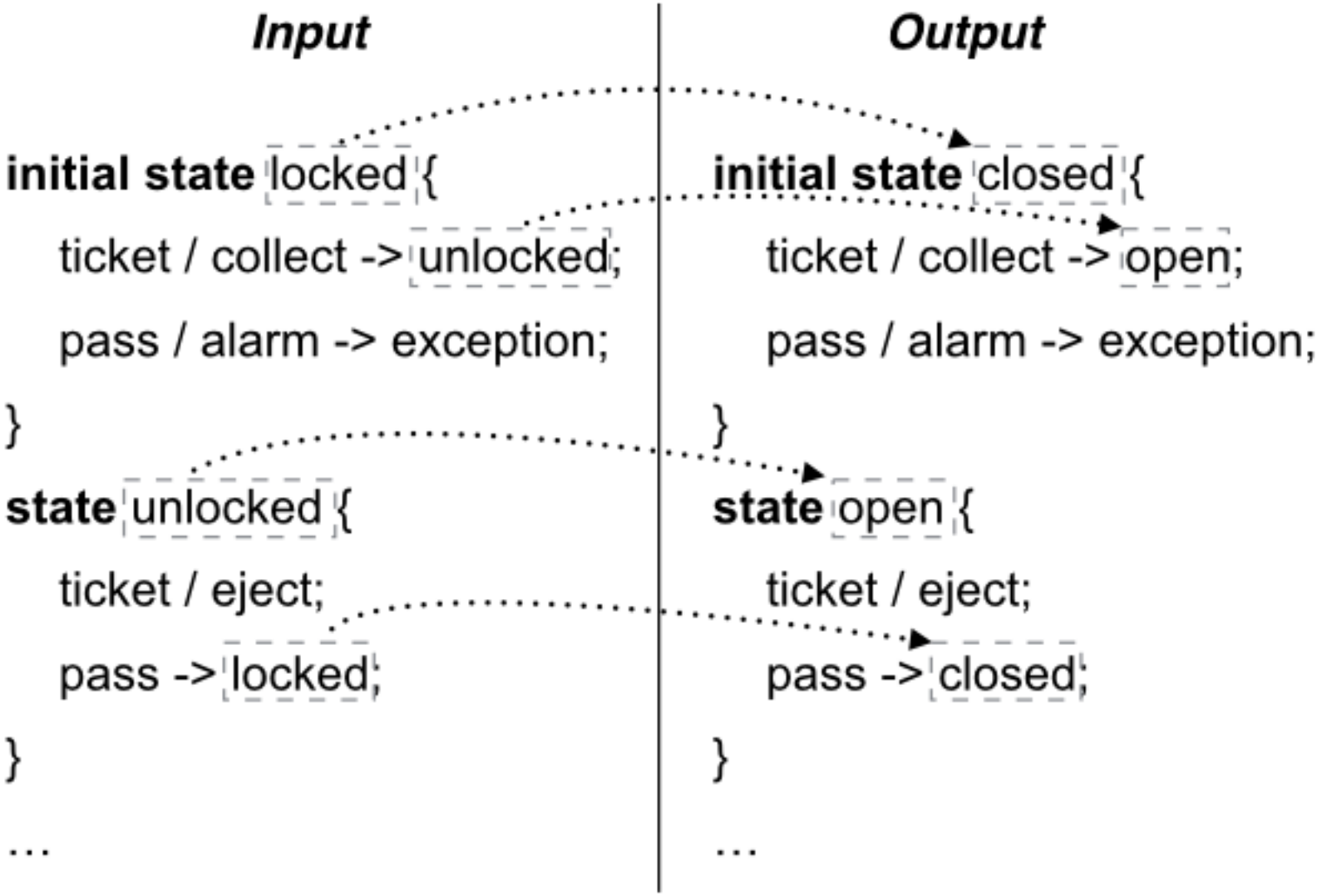


© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Rename refactoring



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Merge composition

```
initial state locked {
    ticket / collect -> unlocked;
}
state unlocked {
    ticket / eject;
    pass -> locked;
}
```

+

```
state locked {
   pass / alarm -> exception;
}
state exception {
   ticket / eject;
   pass;
   mute;
   release -> locked;
}
```

```
initial state locked {
    ticket / collect -> unlocked;
    pass / alarm -> exception;
}
state unlocked {
    ticket / eject;
    pass -> locked;
}
state exception {
   ticket / eject;
   pass;
   mute;
   release -> locked;
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

So let's approach metaprogramming (the comparison of approaches) in a chrestomathic manner.

- Community effort (for aggregation and evaluation)
- Requirement specification
- Multiplicity of languages

chrestomathy

/krɛˈstɒməθi/

noun *formal*

a selection of passages from an author or authors, designed to help in learning a language.

- Rich metadata
- Process management[Google]

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An example of a software chrestomathy

http://rosettacode.org/wiki/Rosetta_Code

Rosetta Code

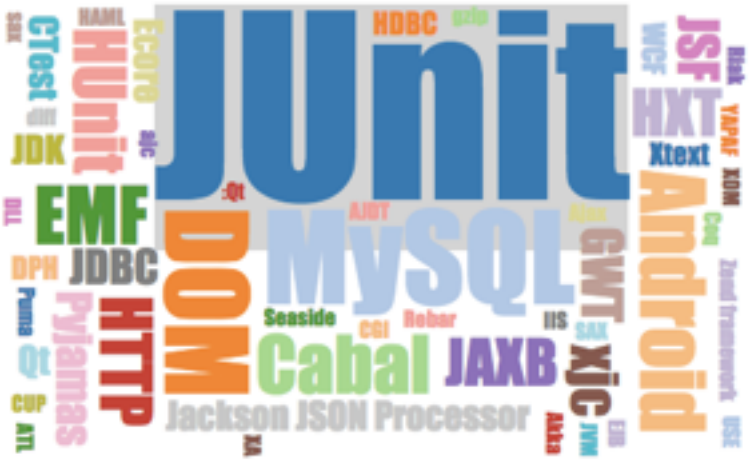
Rosetta Code is a programming chrestomathy site. The idea is to present solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different, and to aid a person with a grounding in one approach to a problem in learning another. Rosetta Code currently has 850 tasks, 198 draft tasks, and is aware of 651 languages, though we do not (and cannot) have solutions to every task in every language.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Another example of a software chrestomathy

https://101wiki.softlang.org/

The project ‘101’ is an open knowledge resource covering software technologies, languages, and concepts. 101 targets programmers, software engineers, teachers, learners, and technologists; they can leverage 101 and they are encouraged to contribute to 101.



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Characteristics of a software chrestomathy

- Community effort (for aggregation and evaluation)
- Requirement specification
- Multiplicity of languages
- Infrastructural support
- Revision and access control
- Quality assurance
- Rich metadata
- Process management

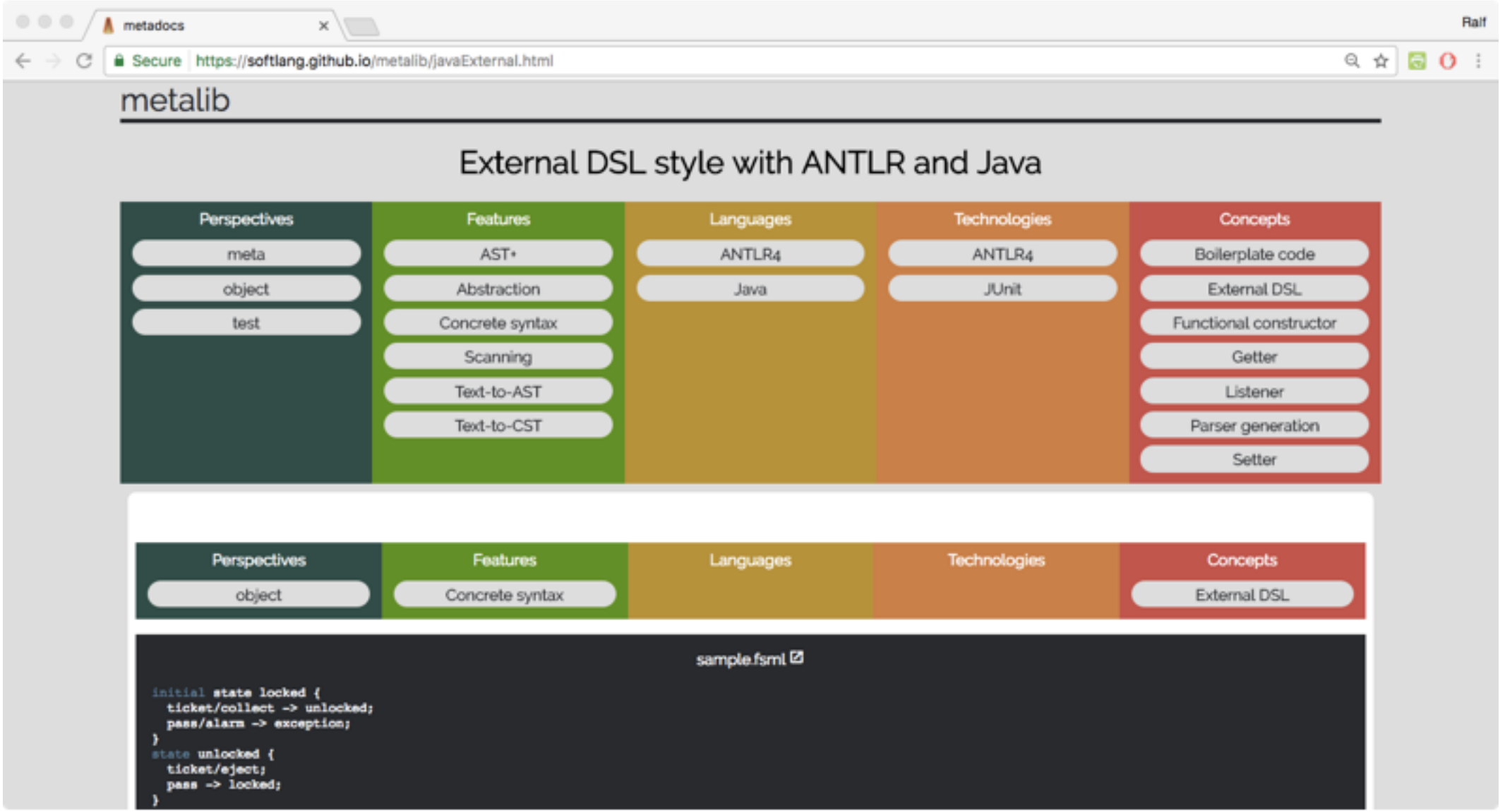
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Chrestomathies related to metaprogramming

We will look at two software chrestomathies that are specifically focused on being useful for learning about software languages at the level of metaprogramming (or language implementation).

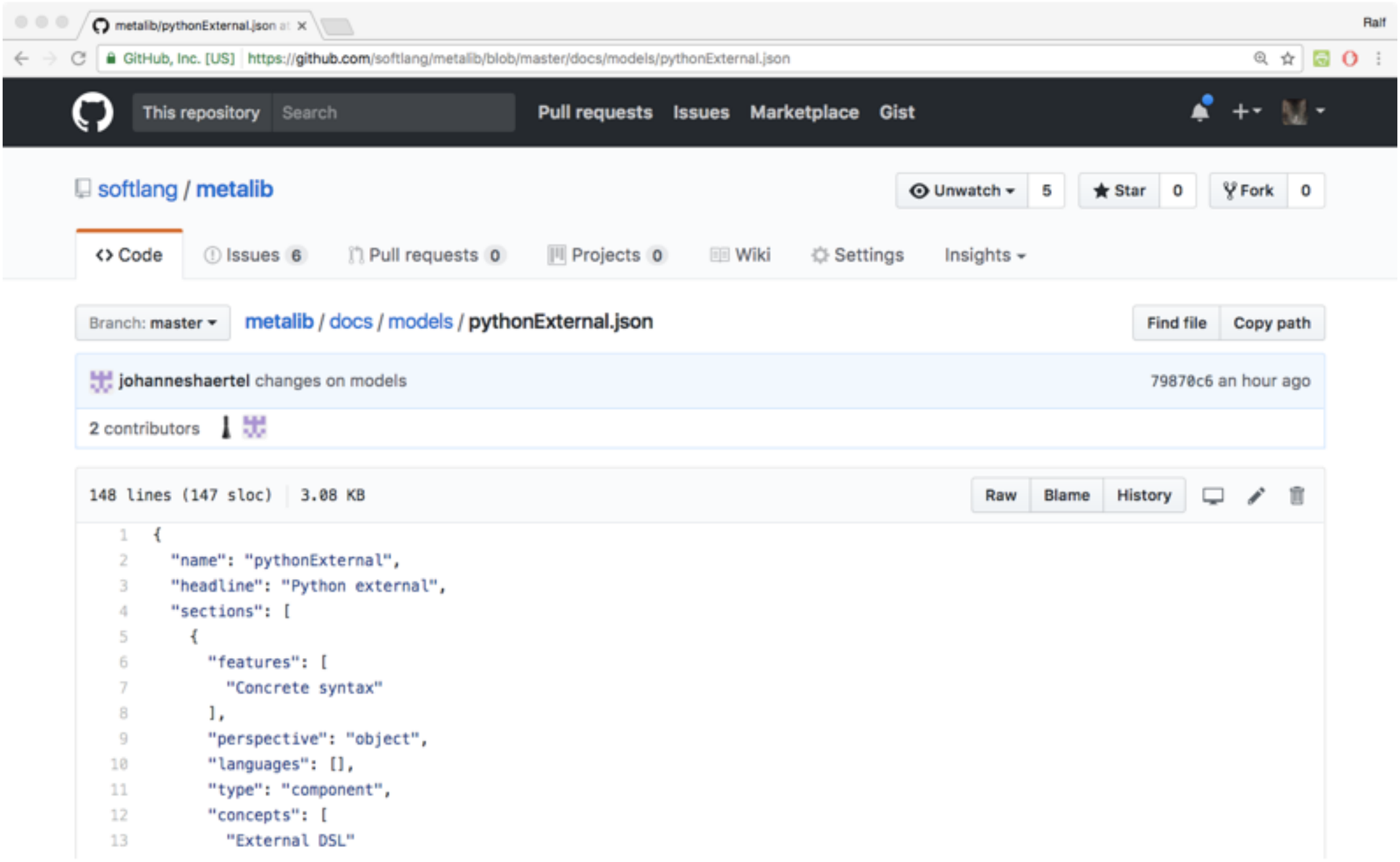
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

MetaLib at https://softlang.github.io/metalib/



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Model-based documentation



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Metamodel of documentation

```
// Documentation of contributions
class document {
  value name : string; // The name of the contribution
  value headline : string; // A one-liner explanation
  value baseuri : string; // Base URI for links
  part sections : section+; // Sections of the documentation
}

// Sections in a documentation
class section {
  value headline : string?; // Optional one-liner explanation
  part perspectives : perspective+; // Perspective of section
  value features : string+; // Features addressed by section
  value languages : string*; // Languages used
  value technologies : string*; // Technologies used
  value concepts : string*; // Concepts used
  part artifacts : artifact+; // Artifacts to be shown
}

// Perspectives of documentation
```

```
// Perspectives of documentation
abstract class perspective { }
// Metaprogram, e.g., grammar or interpreter
class meta extends perspective { }
// Object program in different representations
class object extends perspective { }
// Data other than object programs, e.g., input
class data extends perspective { }
// Build-step for functionality
class build extends perspective { }
// Test for functionality
class test extends perspective { }
// Capture of behavior or appearance
class capture extends perspective { }

// Artifacts for projection
abstract class artifact {
  value link : string; // A relative URI
  value format : string; // MIME-like format type
}
// Nothing to show
class none extends artifact { }
// All to show
```

```
class data extends perspective { }
// Build-step for functionality
class build extends perspective { }
// Test for functionality
class test extends perspective { }
// Capture of behavior or appearance
class capture extends perspective { }

// Artifacts for projection
abstract class artifact {
  value link : string; // A relative URI
  value format : string; // MIME-like format type
}
// Nothing to show
class none extends artifact { }
// All to show
class all extends artifact { }
// A specific line range to show
class some extends artifact {
  value from : integer;
  value to : integer;
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodeling

Acknowledgement: This is joint work with Andrei Varanovich, Marcel Heinz, Johannes Härtel (all SoftLang), and Jean-Marie Favre (University of Grenoble).

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

What's a megamodel?

A model whose model elements are models.

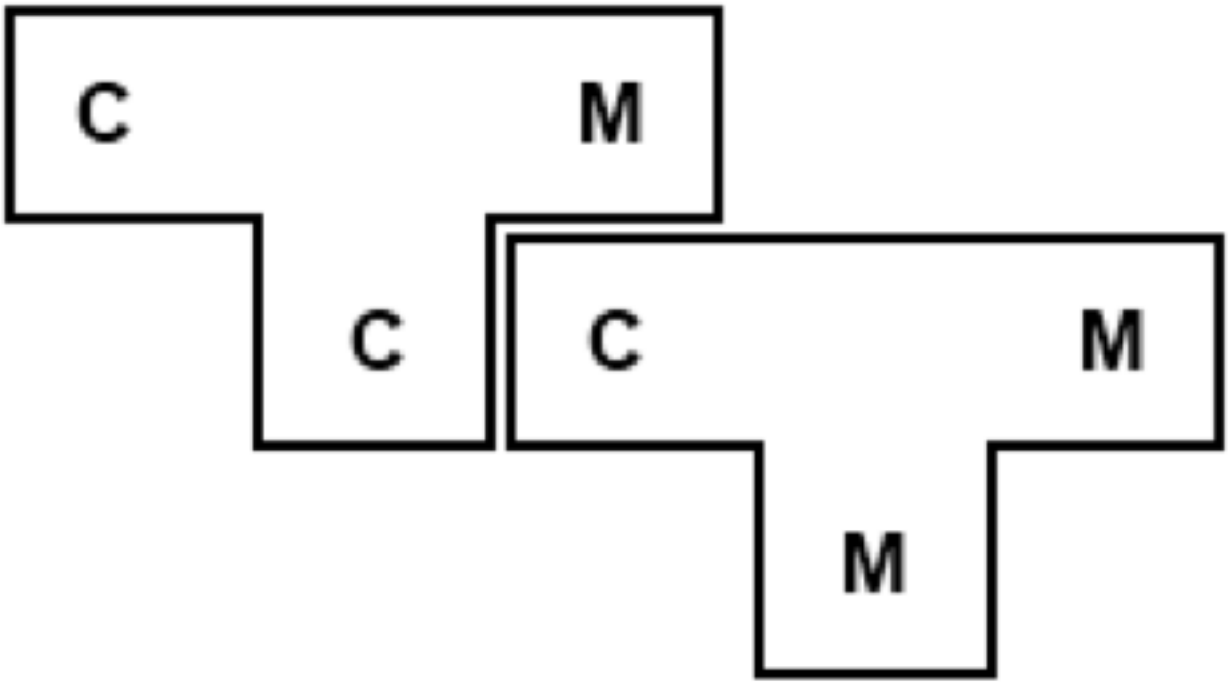
The notion of model is to be interpreted broadly:

- **model, metamodel, model transformation,**
- **program, grammar, metaprogram,**
- **document, schema, transformation,**
- **…**

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An example of a megamodel:

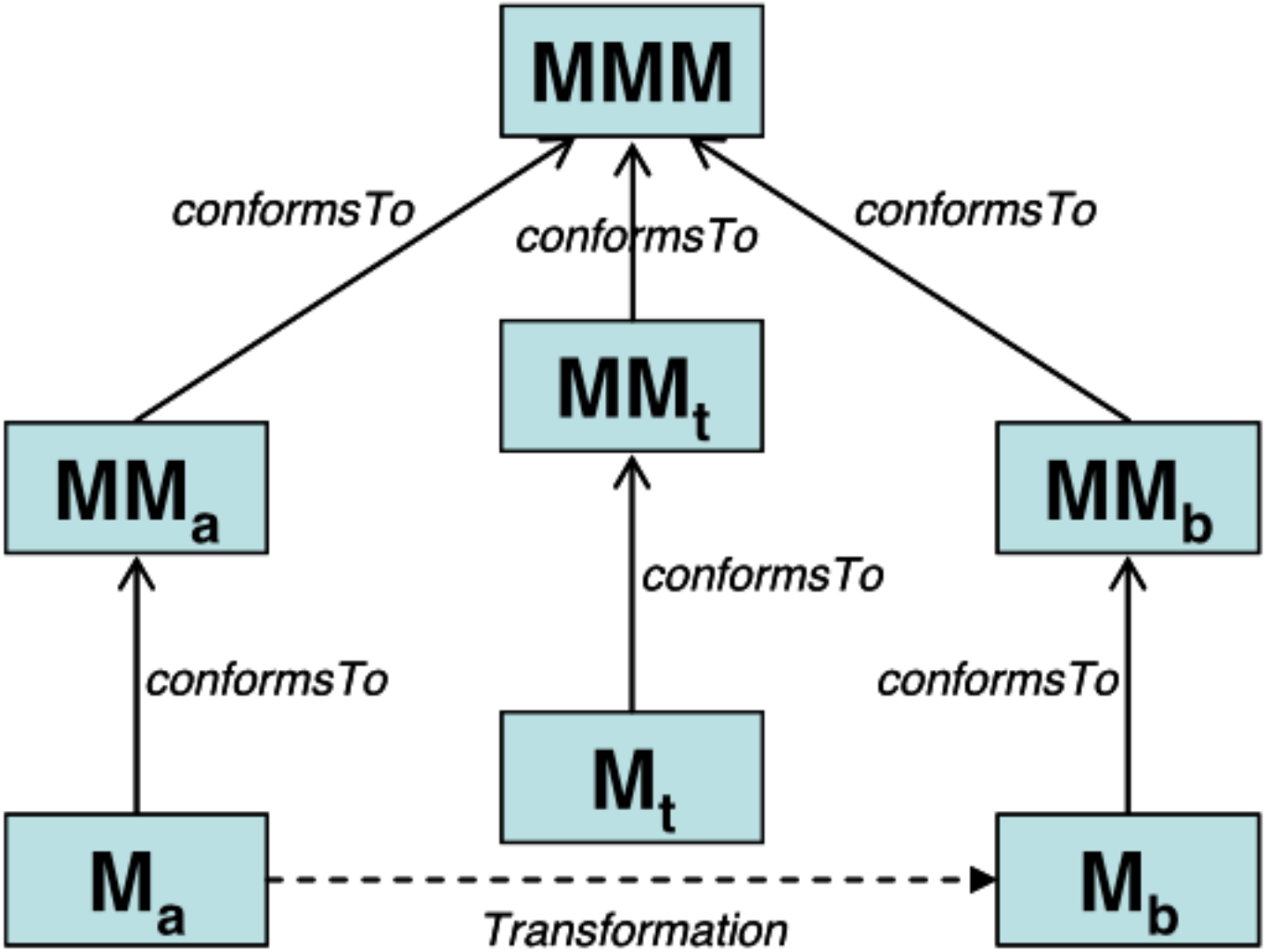
A tombstone diagram for bootstrapping a compiler



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Another example of a megamodel:

Mechanics of an ATL-based transformation



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Technology modeling

(as a form of megamodeling)

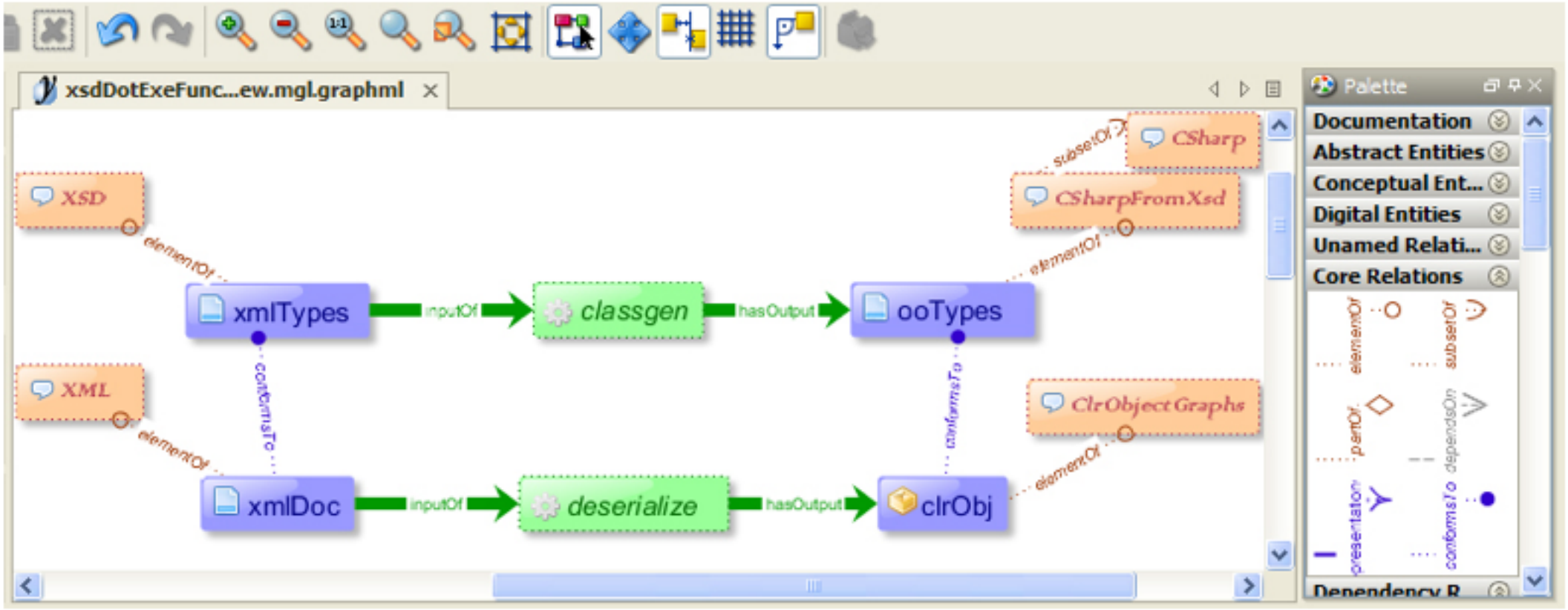
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Basic terminology

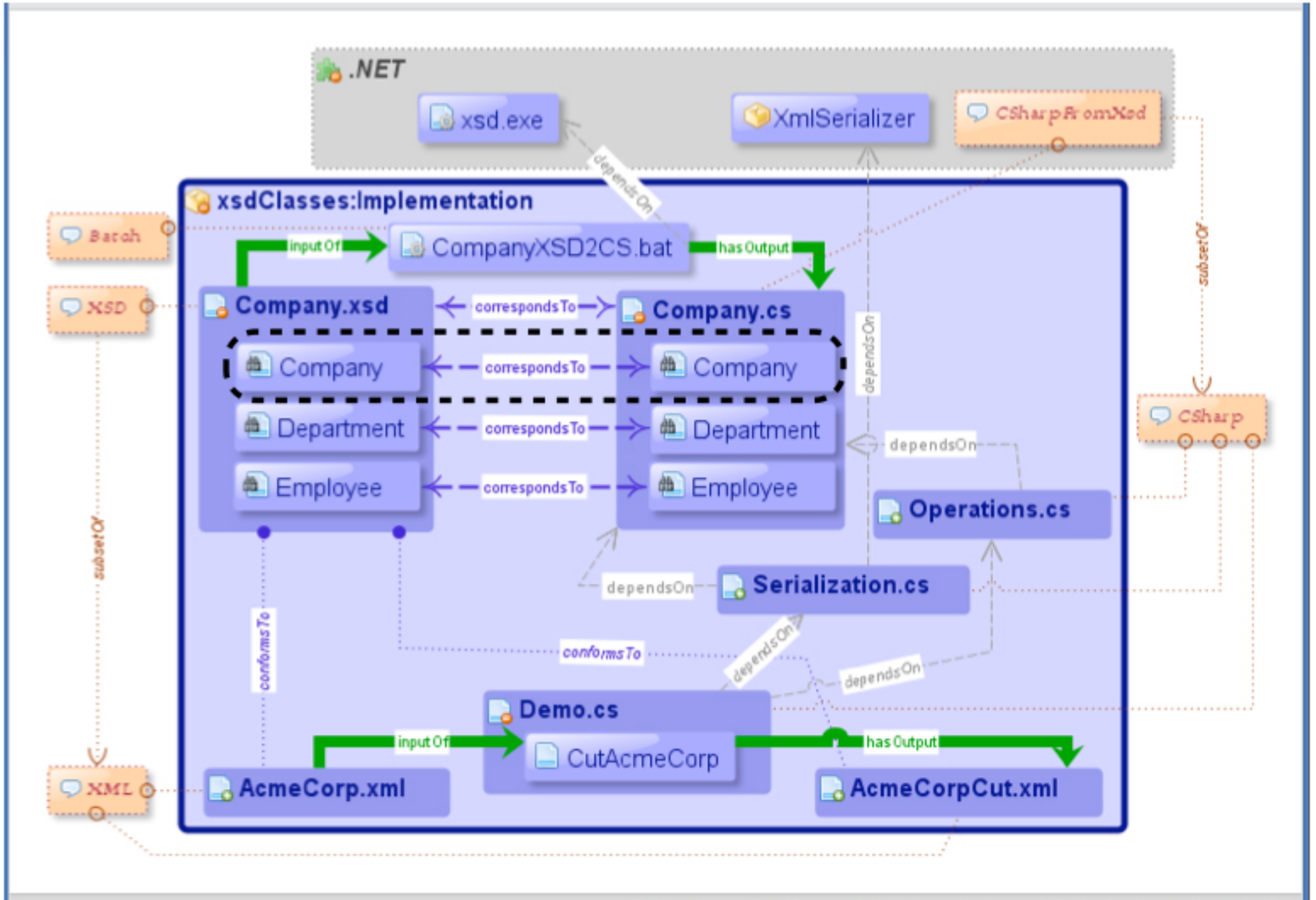
- Technology = Software technology, e.g.:
  - Web-application framework
  - O/R mapper
- Technology modeling:
  - an **ontological documentation** of technology
  - a form of **megamodeling**
  - also referred to as (a model of) **linguistic architecture**

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Linguistic architecture of xsd.exe

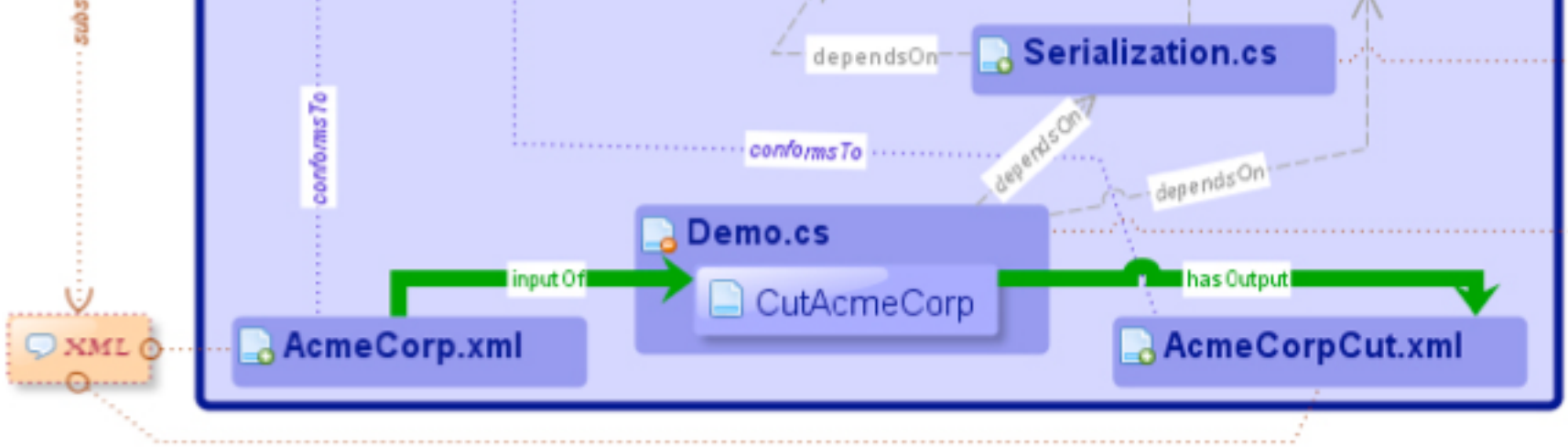


© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.




```
1.  <?xml version="1.0" encoding="UTF-8" star
2.  <xs:schema xmlns="http://www.softlang.org
3.
```

```
21.  [System.Diagnostics.DebuggerStepThroughAt
22.  [System.ComponentModel.DesignerCategoryAt
23.  [System.Xml.Serialization.XmlTypeAttribut
24.  [System.Xml.Serialization.XmlRootAttribut
```




```
1.  <?xml version="1.0" encoding="UTF-8" star
2.  <xs:schema xmlns="http://www.softlang.org
3.
4.    <xs:element name="Company">
5.      <xs:complexType>
6.        <xs:sequence>
7.          <xs:element name="Name" type="xs:
8.          <xs:element name="TopLevelDepartm
9.        </xs:sequence>
10.     </xs:complexType>
11.   </xs:element>
12.
13.   <xs:complexType name="Department">
14.     <xs:sequence>
15.       <xs:element name="Name" type="xs:st
16.       <xs:element name="Manager" type="En
17.       <xs:element name="SubDepartment" ty
18.       <xs:element name="Employee" type="E
19.     </xs:sequence>
20.   </xs:complexType>
```

```
21.  [System.Diagnostics.DebuggerStepThroughAt
22.  [System.ComponentModel.DesignerCategoryAt
23.  [System.Xml.Serialization.XmlTypeAttribut
24.  [System.Xml.Serialization.XmlRootAttribut
25.  public partial class Company {
26.
27.      private string nameField;
28.
29.      private Department[] topLevelDepartme
30.
31.      /// <remarks/>
32.      public string Name {
33.          get {
34.              return this.nameField;
35.          }
36.          set {
37.              this.nameField = value;
38.          }
39.      }
40.
41.      /// <remarks/>
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Overarching research questions

- What are problems with *classic documentation*?
- How to ontologically structure and enrich documentation?
- What is the underlying *vocabulary* and *ontology*?
- How to create *value for developers* (like Stackoverflow)?
- What sort of *tool support* is necessary or helpful?
- How to actually *renarrate* such documentation?

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A technology model

for using EMF+Xtext+ATL

for the purpose of DSML implementation

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodel of the XML story

```
module XML import (Prelude) // Import basic vocabulary
XML : Language // Declare XML as a language entity
XSD : Language // and XSD (XML Schema), too
XSD subsetOf XML // Subset relationship on XSD and XML
xmlFile : Artifact // Declare artifact
xsdFiles : Artifact+ // Declare collection of artifacts
xmlFile elementOf XML // Assign language to artifact
xsdFiles elementOf XSD // Assign language to artifact
xmlFile conformsTo xsdFiles // XSD—based validation
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodel of the EMF story

```
module EMF import (Prelude)
Ecore : Language // As defined by metaMetaModel
Custom : Language // As defined by metaModel
metaModel : Artifact // A metamodel artifact
metaMetaModel : Artifact // The metametamodel
metaModel elementOf Ecore
metaMetaModel elementOf Ecore
metaModel conformsTo metaMetaModel
metaMetaModel conformsTo metaMetaModel
metaModel defines Custom
metaMetaModel defines Ecore
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodel of the ATL story

module ATL import (EMF) // *ATL depends on EMF*
transformation : Custom ⟶ Custom // *Function on DSL*
input : Artifact // *Input artifact*
output : Artifact // *Output artifact*
input elementOf Custom // *input (source) of transformation*
output elementOf Custom // *output (target) of transformation*
transformation(input) ⟼ output // *Function application*
ATL : Language // *The ATL language*
atlmodule : Artifact // *An ATL transformation module*
atlmodule elementOf ATL
atlmodule defines transformation // *Semantics of ATL module*

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodel of the Xtext story

```
module Xtext import (EMF) // Xtext integrates with EMF
Xtext : Language // Xtext language
grammar : Artifact // An artifact for the grammar
grammar elementOf Xtext // An Xtext grammar
EcoreWithoutOps : Language // Relevant subset of Ecore
EcoreWithoutOps subsetOf Ecore
metaModel elementOf EcoreWithoutOps // Restriction of import
metaModel correspondsTo grammar // Correspondence
generator : Xtext ⟶ EcoreWithoutOps // Generator function
generator(grammar) ⟼ metaModel // Generator application
MWE2 : Language // Language for generator configuration
workflow : Artifact // Workflow artifact
workflow elementOf MWE2 // Workflow is written in MWE2
workflow defines generator // Workflow defines generator function
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodel of EMF Model API story

```
module EMFModelAPI import (
   EMF, // The EMF module is enhanced
   XML) // The XML module is needed for serialization
Java : Language // Java is a Language
EcoreJava : Language // A Java subset for EMF Model APIs
EcoreJava subsetOf Java // An EMF Model API is valid Java
EMFGenModel : Language // Language for the generator model
genModel : Artifact // Parameters of the generation
genModel elementOf EMFGenModel
genModel references metaModel // Referencing
EMFGenerator : EMFGenModel → EcoreJava
EMFGenerator(genModel) ↦ javaFiles // Application of generator
CustomObjects : Language // Object graphs for Custom
Serialization : CustomObjects → Custom
Deserialization : Custom → CustomObjects
XMI : Language // Format for default persistence for EMF
```

XMI : Language // *Format for default persistence for EMF*
XMI subsetOf XML // *XMI is a subset of XML*
Custom subsetOf XMI // *Custom uses default persistence*
javaFiles : Artifact+ // *The modeled/defined API*
javaFiles elementOf EcoreJava
metaModel correspondsTo javaFiles // *Close resemblance*
javaFiles defines CustomObjects
javaFiles defines CustomSerialize
javaFiles defines CustomDeserialize
model : Artifact // *A serialized artifact*
model elementOf Custom // *... of Custom language*
model conformsTo metaModel // *Conformance to metamodel*
objectGraph : Transient // *A runtime artifact*
objectGraph elementOf CustomObjects
objectGraph conformsTo javaFiles // *Conformance to Java classes*
CustomSerialize(objectGraph) $\mapsto$ model
CustomDeserialize(model) $\mapsto$ objectGraph

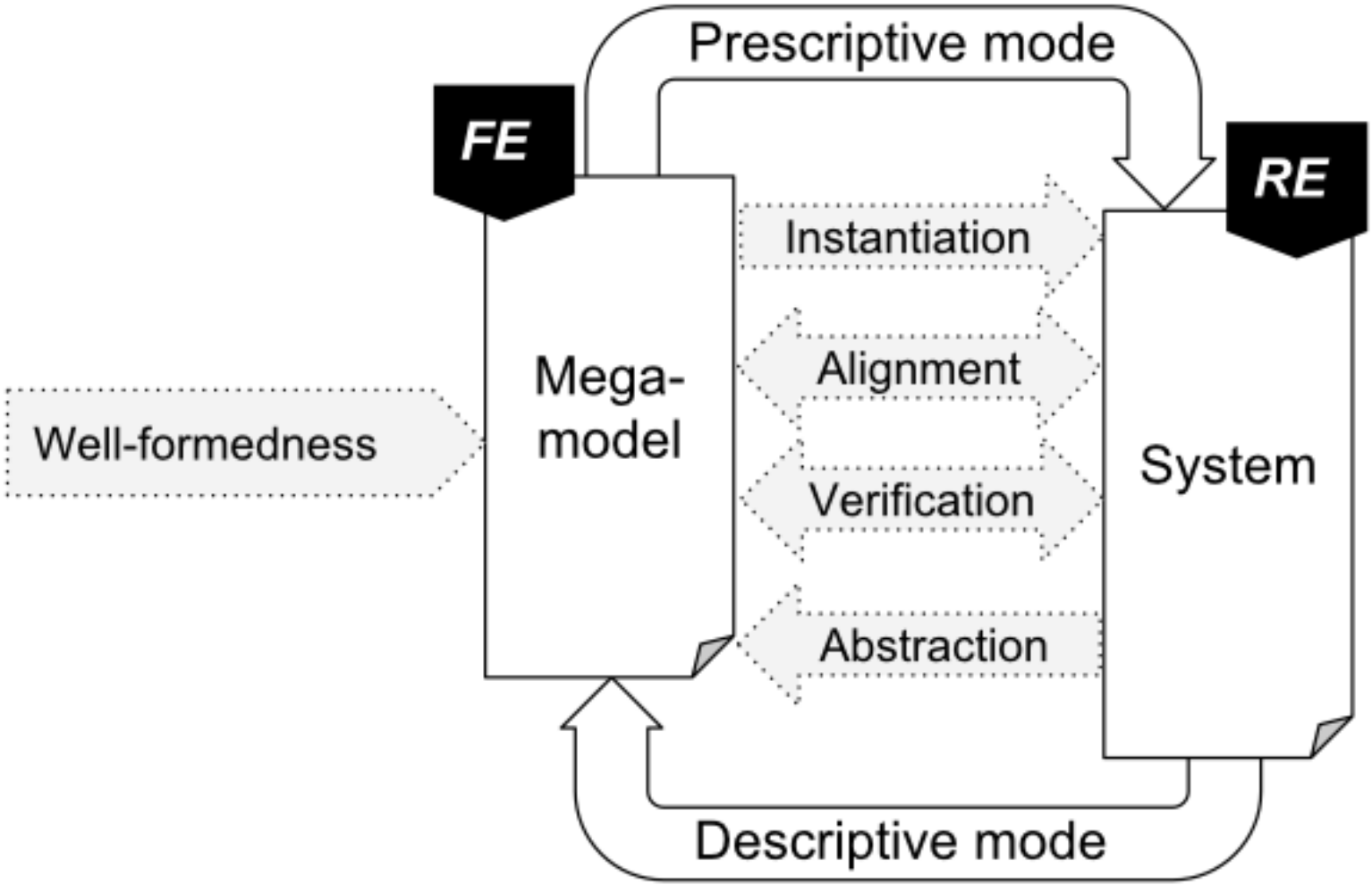
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Discovery of entities and relationships

Id	Question	Relevant MegaL constructs
L1	Which languages can be identified?	Type *Language*
L2	Is one language contained in another?	Relationship *subsetOf*
A1	What artifacts participate in the scenario?	Type *Artifact*
A2	What is the language of each artifact?	Relationship *elementOf*
A3	Does an artifact conform to another artifact?	Relationship *conformsTo*
A4	Does an artifact define a language?	Relationship *defines*
F1	Is one artifact derived from another artifact?	Type *Function*
F2	What is domain and range of a function?	Function with domain & range
F3	How is a function applied?	Function application '$f(x) \mapsto y$'
F4	How is a function defined?	Relationship *defines*
R1	Are artifacts closely similar to each other?	Relationship *correspondsTo*
R2	Can a correspondence be structured?	Relationship *partOf*
R3	What causes a correspondence?	Function application '$f(x) \mapsto y$'
C1	Can the entity be described conceptually?	Type *Concept*
C2	Does the entity use the concept?	Relationship *uses*
C3	Does the entity help to use the concept	Relationship *facilitates*

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

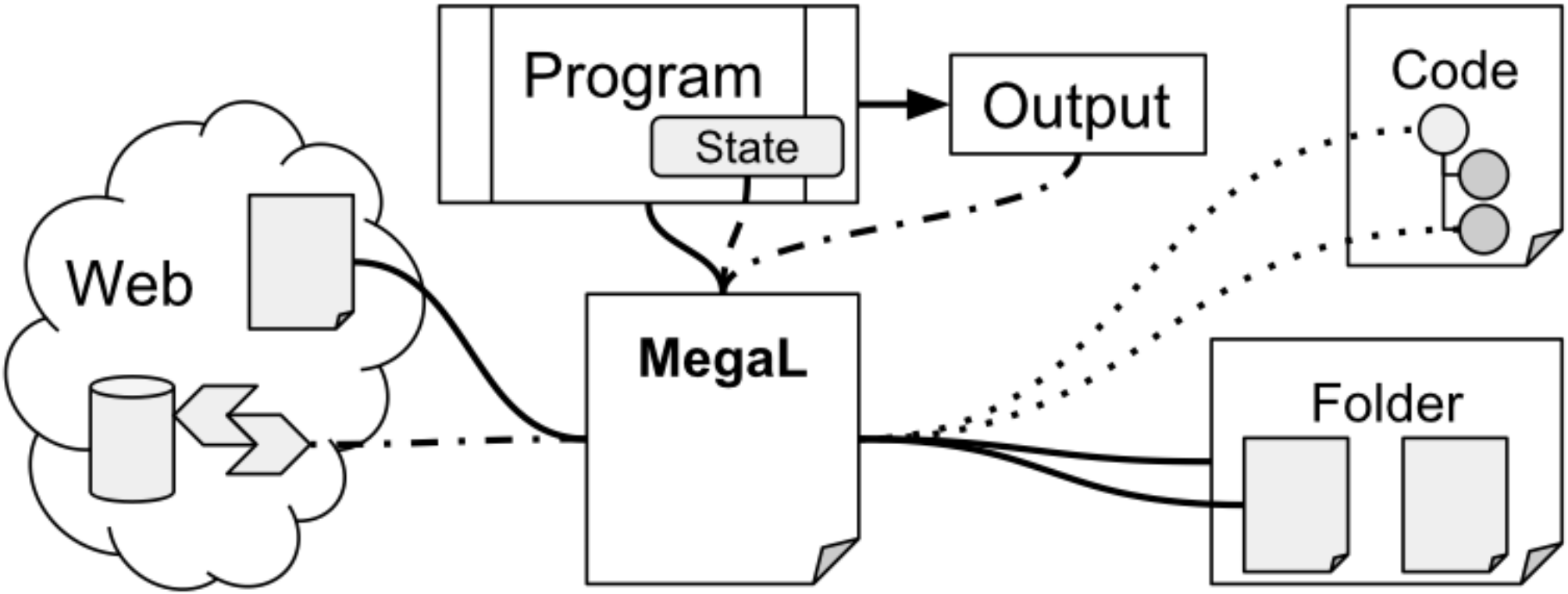
Linguistic architecture in forward and reverse engineering



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Interconnected linguistic architecture

(as an advanced form of modeling
that emphasizes connection between model
and system and other elements)



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

URI-based resolution

`github://user/project/files/data.jar/content.xml/root/models/model#1`

- GitHub repo
- File
- Archive content
- File again
- XML-based (XPath-like) selection

```
// module EMF continued
metaMetaModel = 'eclipse:/org.eclipse.emf.ecore/model/Ecore.ecore'
```

MegaL

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Semantic annotations

‘Identity’ links to Wikipedia etc.

XML	http://dbpedia.org/page/XML
EMF	https://eclipse.org/modeling/emf/

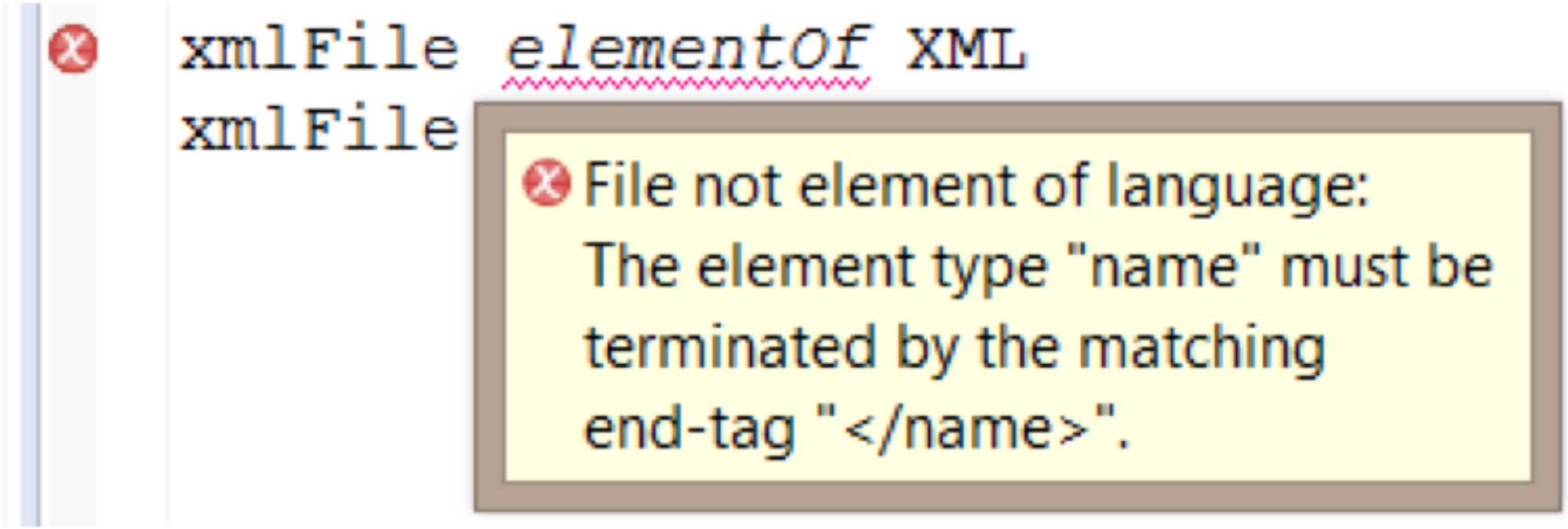
```
// module XML continued
XML = 'http://dbpedia.org/page/XML'
```

MegaL

```
// module EMFModelAPI continued
Persistence : Concept
Persistence =
    'http://dbpedia.org/page/Persistence_(computer_science)'
CustomSerialize facilitates Persistence
CustomDeserialize facilitates Persistence
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Pluggable analyses



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

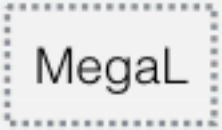
Plugin

```
class XMLConformsToXSD extends MegaLEvaluator {
   // Returns an evaluation report on the model element
   protected Report<Void> evaluate(Relationship element) {
      // Use SAX for validation; translate exceptions to report
      ...
   }
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

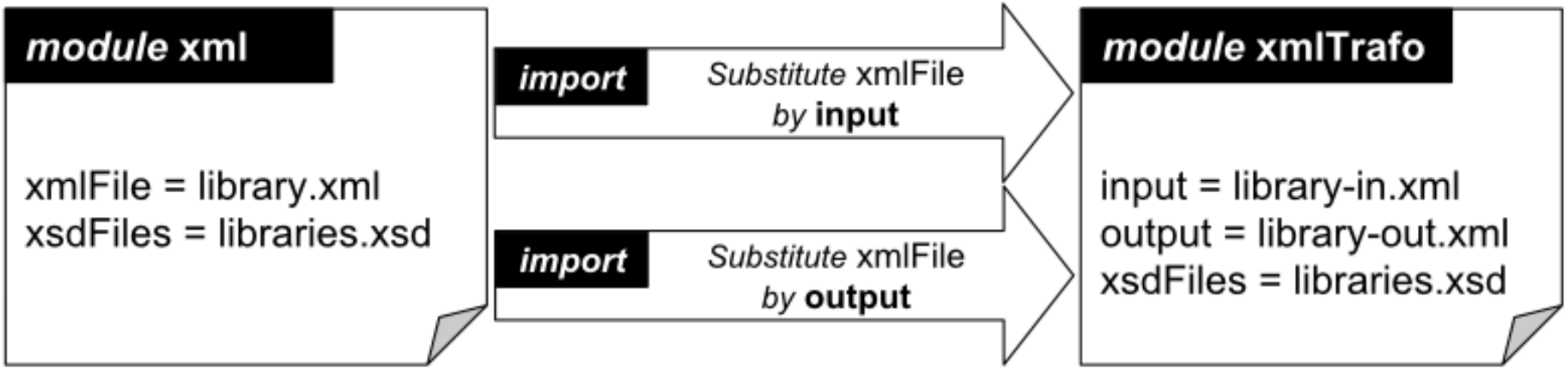
Plugging in

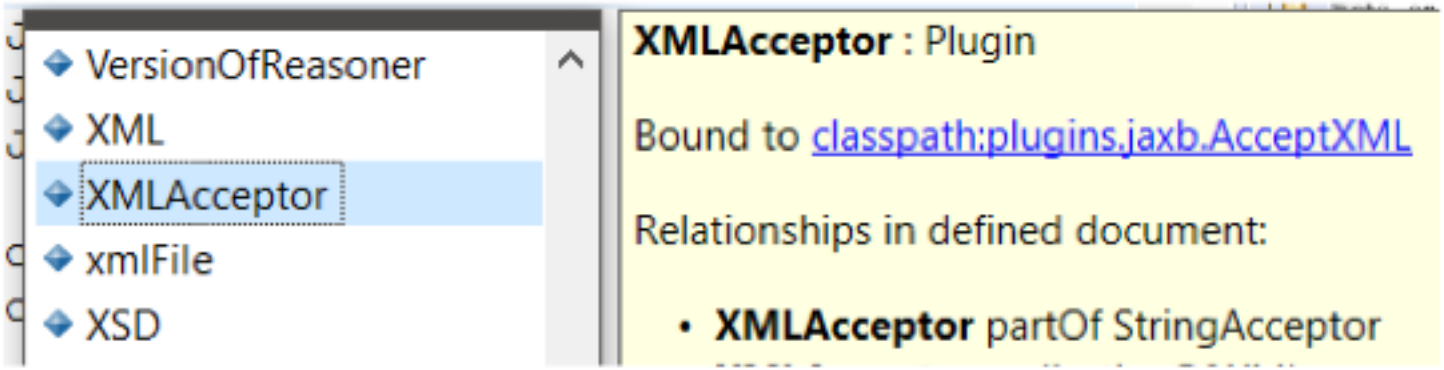
```
conformsTo < Artifact * Artifact // Relationship type per prelude
ConformsToEvaluator : Plugin // Root plugin for conformance
ConformsToEvaluator = "classpath:ConformsToEvaluator"
conformsTo evaluatedBy ConformsToEvaluator
XMLConformsToXSD : Plugin // XML/XSD conformance
XMLConformsToXSD = "classpath:XMLConformsToXSD"
XMLConformsToXSD partOf ConformsToEvaluator
SAX : Technology // Semantic annotation of plugin
SAX = 'http://dbpedia.org/page/Simple_API_for_XML'
XMLConformsToXSD uses SAX
```



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

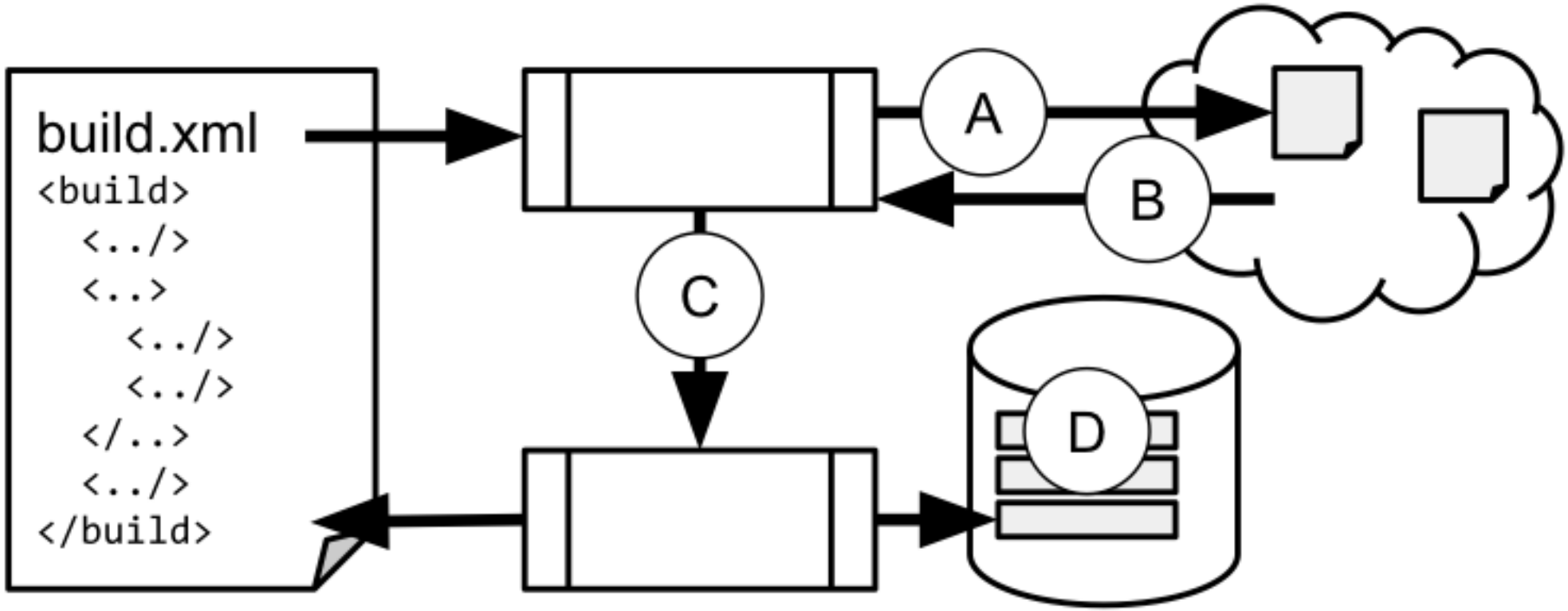
Modularized models





© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Transient artifacts



- A and B: web request and response
- C: piped program output
- D: transient data in memory or database

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

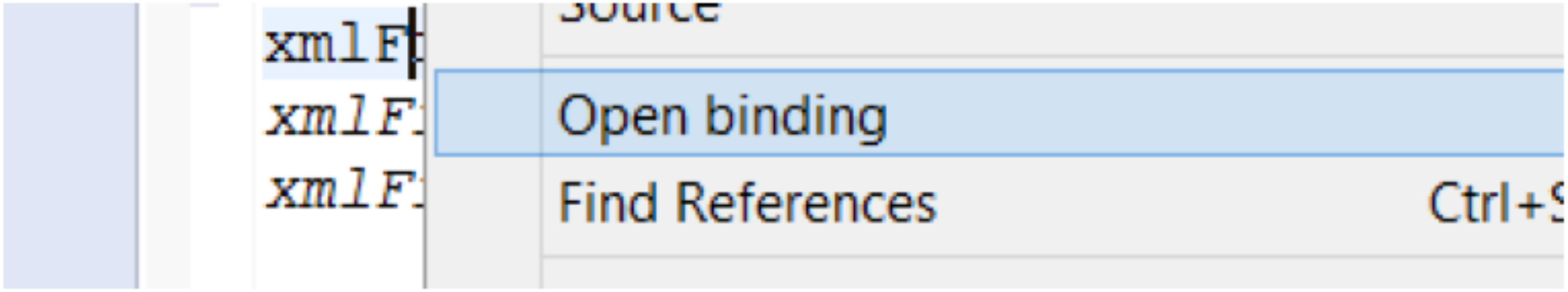
Model inference

- Decomposition in parts
- Correspondence between parts
- Artifact bindings

```
class EMFPartInferrer extends MegaLInferrer {
  // Returns an evaluation report and a model extension
  protected Report<Megamodel> infer(Entity element) { ... }
}
```

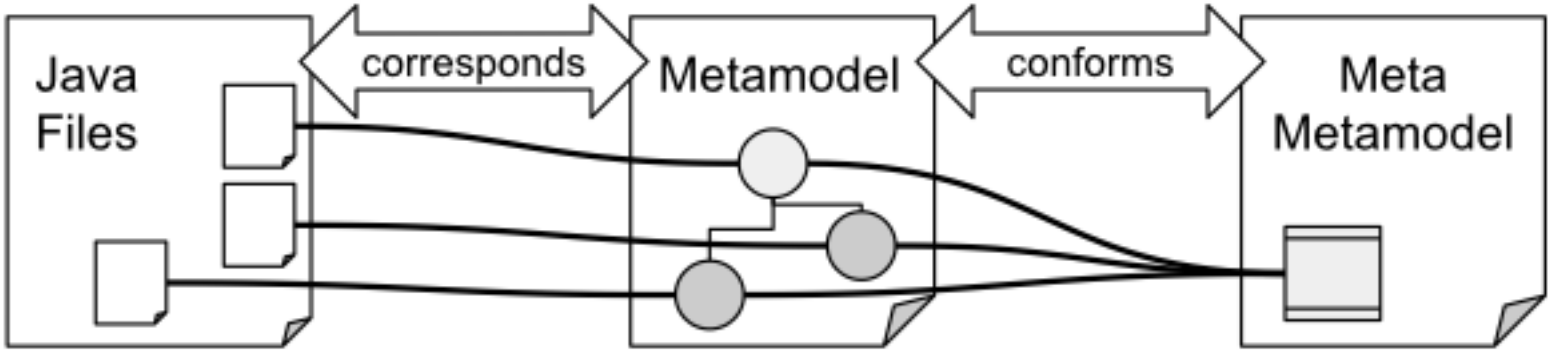
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Explorable connections



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Traceability links



xsdFiles	javaFiles
/xs:schema/xs:complexType	org/softlang/company/xjc/Employee.java
/xs:schema/xs:element#0	org/softlang/company/xjc/Company.java
/xs:schema/xs:element#1	org/softlang/company/xjc/Department.java
xmlFile	objectGraph
company/department#0	org.softlang.company.xjc.Department@5fd1a6aa
employee#0	org.softlang.company.xjc.Employee@1a56a6c6
address:Utrecht	Utrecht
name:Erik	Erik
salary:12345	12345.0
employee#1	org.softlang.company.xjc.Employee@748e432b

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

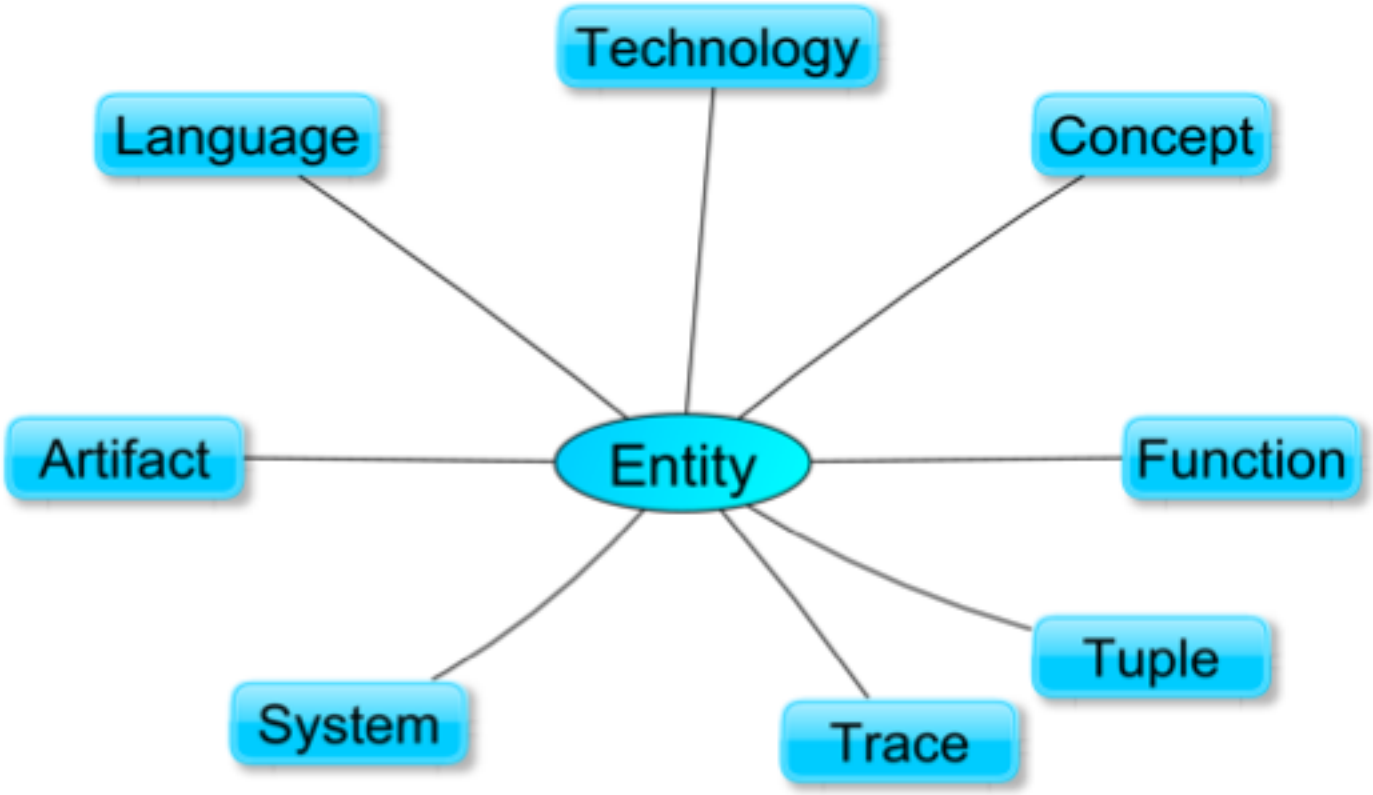
Literature survey

	Traceability links	Artifact binding	Model inference	Pluggable analyses	Explorable connections	Modularized models	Semantic annotations	Transient artifacts
[30]		·		•		•		·
[7]	•	·		·				
[25]	•	•	•	•		·		
[15]	·			·		·		
[11]	•	·			·			
[17]	•	·	•		·			
[12]	•	·	•	·				
[3]	•	·	•		·			
[4]	•	•	•	·	·			
([10])	•	•	·	·	•		•	
L_3		○		○			○	
L_2			⊗	×	×	×		○
L_1	⊗	×			○	○	×	×
	3.8	3.1	3.6	3.3	3.7	3.4	3.2	3.5

- L_1-L_3 maturity levels
- **O** implementation
- × demonstration

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Axioms of linguistic architecture



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Entity types in megamodeling survey

Paper	Artifact	Function	Record	System	Technology	Language	Inf. resource	Fragment	Collection	Trace	Concept	Others
[1]	x	x	x				x					x
[2]	x	x	x		x					x		x
[3]	x			x	x						x	x
[4]					x	x	x				x	x
[5]	x						x	x		x		x
[6]	x		x									
[7]	x	x	x									
[8]	x									x		
[9]	x						x		x			
[10]	**x**	**x**	x		x	x		x	x			
[11]	x	x	x							x		
[12]				x								x
[13]	x	x								x		x
[14]	x	x										

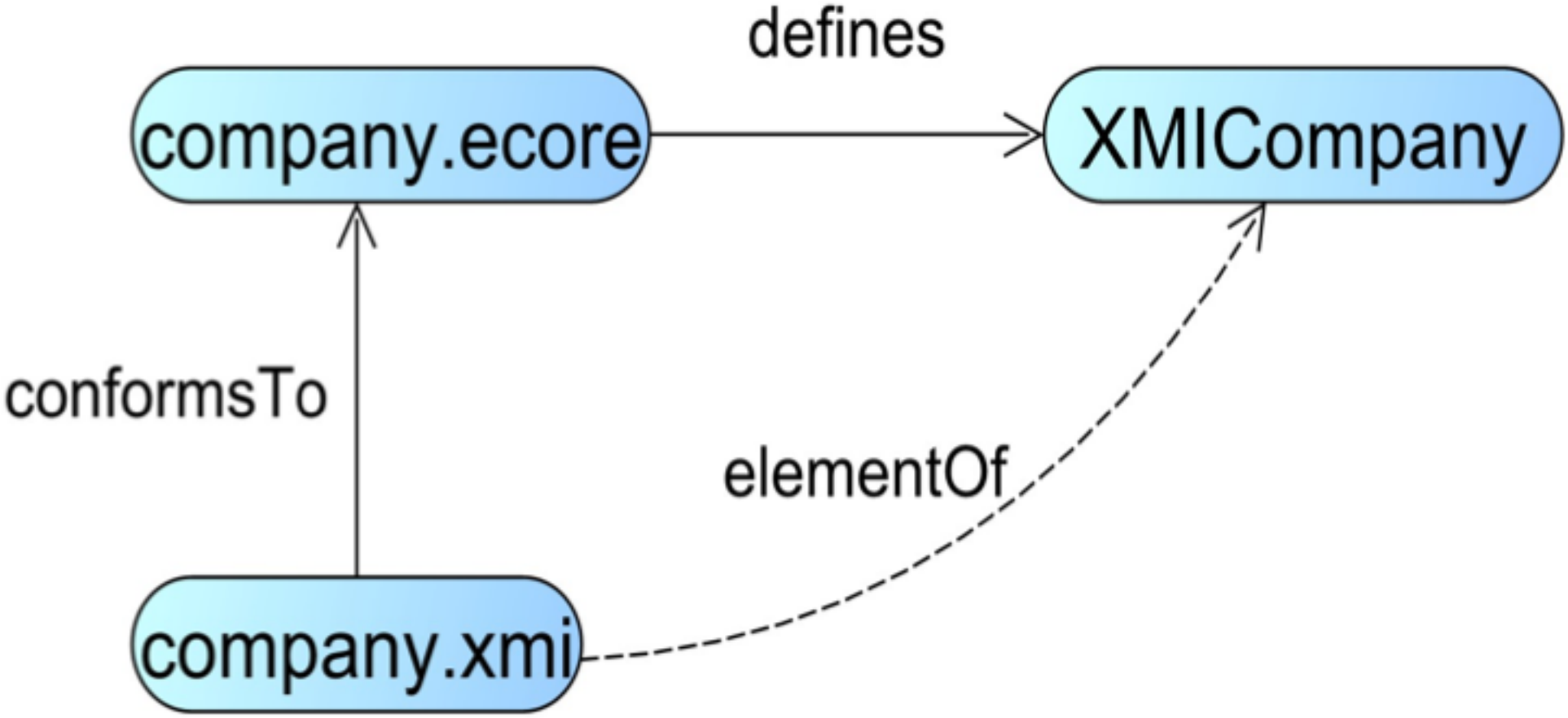
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Relationship types in megamodeling survey

Paper	Conformance	Definition	Correspondence	Implementation	Usage	Membership	Typing	Dependency	Abstract rel.	Others
[1]					x					x
[2]	x									
[3]	x	x	x	x	x			x		x
[4]				x	x		x	x		x
[5]				x			x			x
[6]						x	x		x	
[7]										x
[8]									x	
[9]		x							x	
[10]	x	x	x	x		x	x	x	x	
[11]	x	x					x		x	
[12]		x								x
[13]							x			
[14]	x								x	

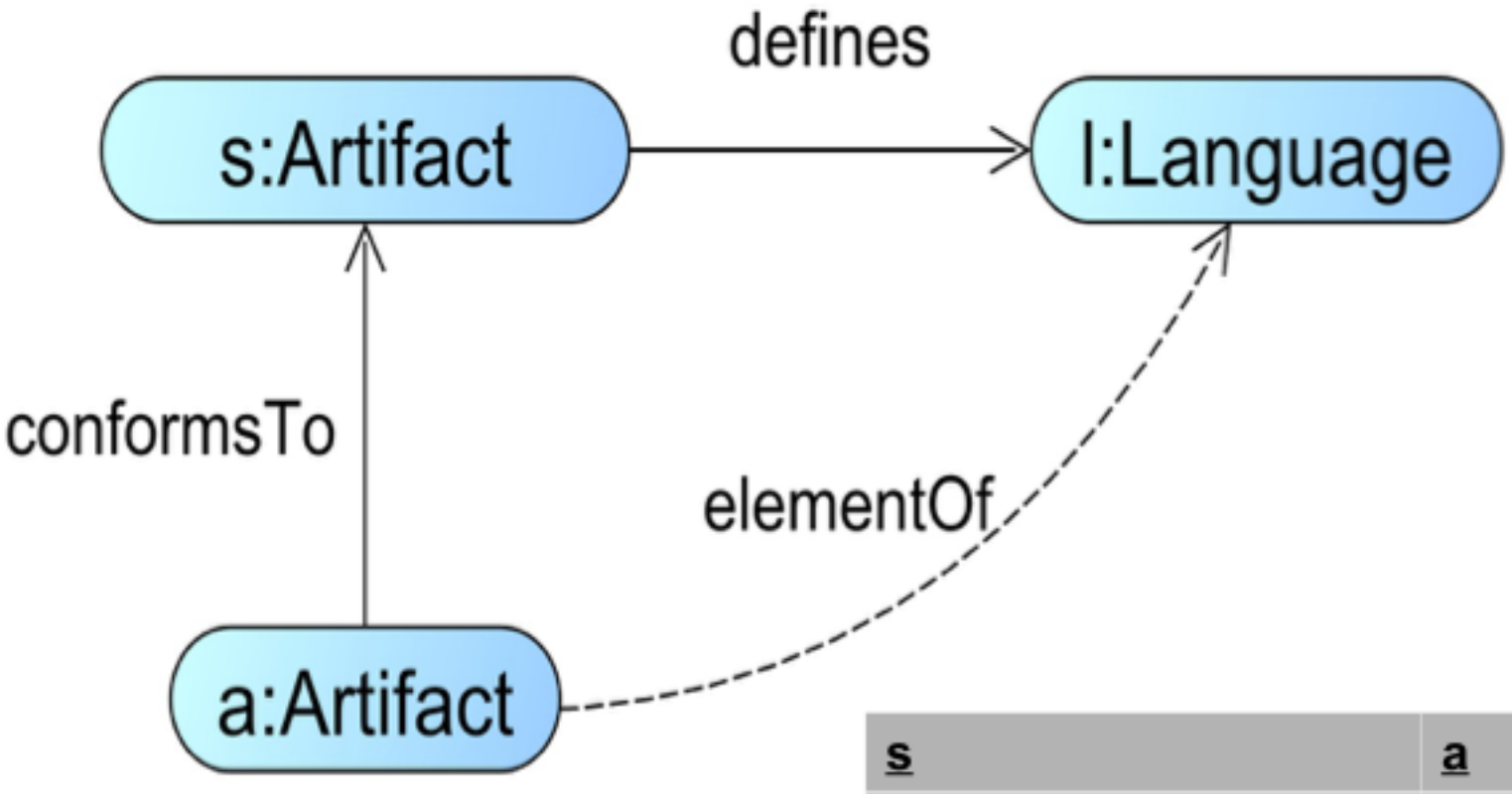
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Understanding Membership



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

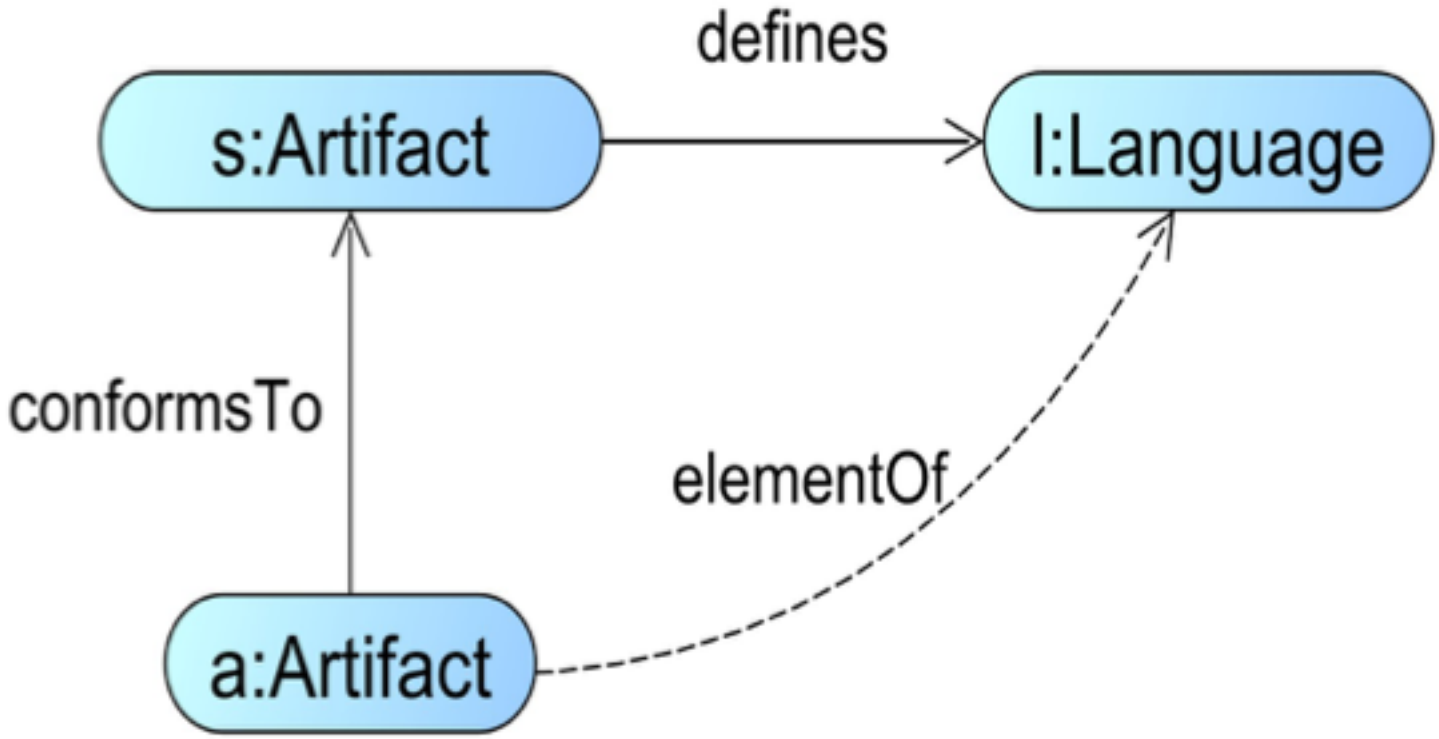
Understanding Membership



s	**a**
Grammar	Code
Schema	Instance
Metamodel	Model

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Understanding Membership



- $\text{elementOf}(a, l) \Rightarrow \text{Artifact}(a) \wedge \text{Language}(l)...$
- $\text{elementOf}(a, l) \Leftarrow \exists s.\text{defines}(s, l) \wedge \text{conformsTo}(a, s).$

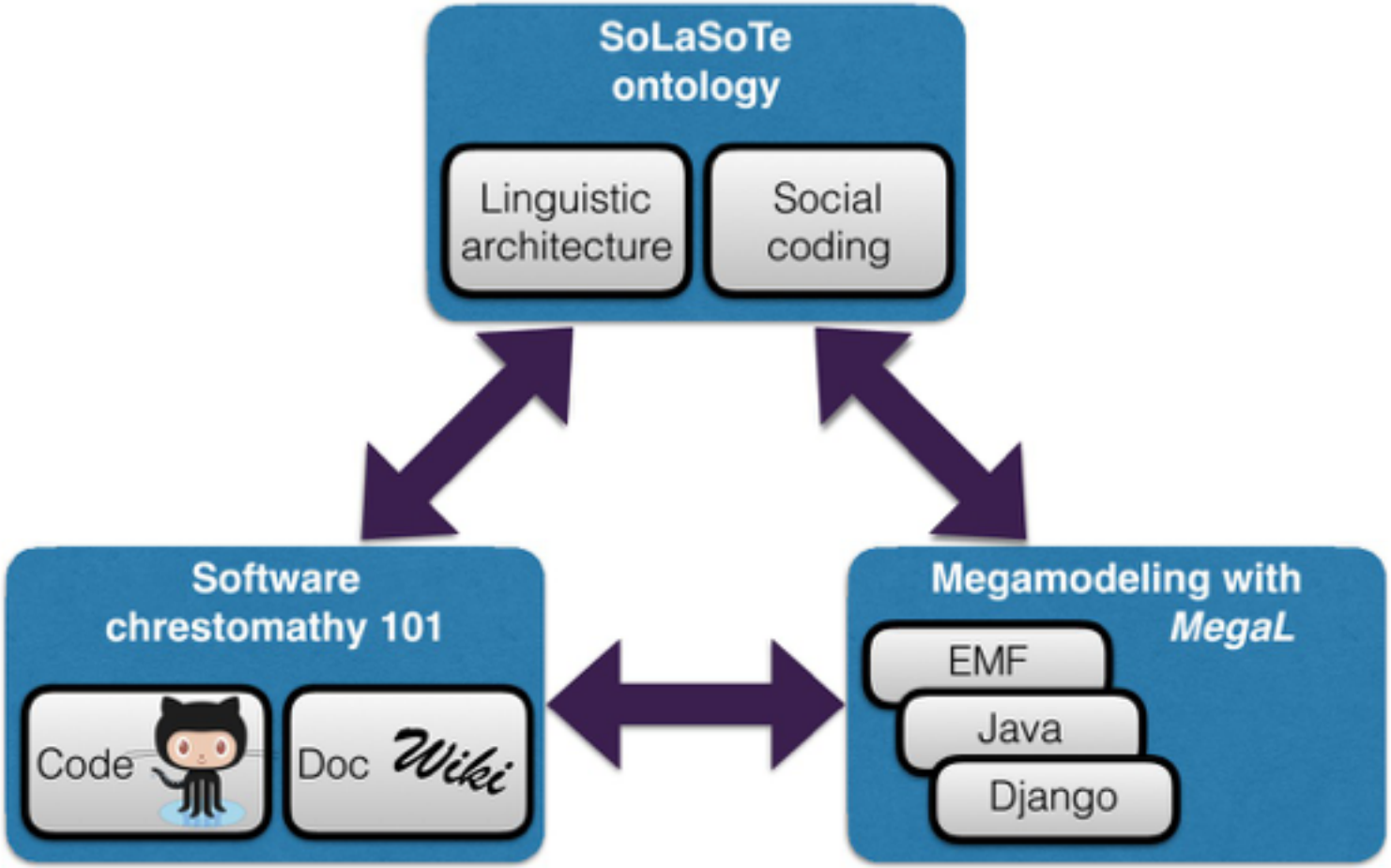
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Understanding Membership

- $\text{Specification}(a) \Rightarrow \text{Artifact}(a)$.
- $\text{Language}(l) \Rightarrow \exists s.\text{Specification}(s) \wedge \text{defines}(s, l)$ ...
- $\text{defines}(a, e) \Rightarrow \text{Artifact}(a) \wedge \text{Entity}(e)$.
- $\text{conformsTo}(a, s) \Rightarrow \text{Artifact}(a) \wedge \text{Artifact}(s)$.
- $\text{conformsTo}(a, s) \Leftarrow (\forall p_a.\text{partOf}(p_a, a) \wedge \exists p_s.\text{partOf}(p_s, s) \wedge \text{conformsTo}(p_a, p_s)) \vee \exists t.\text{defines}(s, t) \wedge \text{elementOf}(a, t)$.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Ontology engineering



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Megamodeling

for repository management

(proxying for build management and regression testing)

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

YAS (Yet Another SLR (Software Language Repository))

- 107 languages. (This includes different representation types.)
- 558 language-typed artifacts.
- 121 language-typed functions.
- 391 function applications.
- 252 Prolog modules.
- 171 Haskell modules.
- 111 Java classes.
- 19 Python scripts.

How to build and test such a repository?
How to understand all the dependencies?

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Illustrative YAS artifacts

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A binary number

101.01

BTW, everything is a file in a/this SLR.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

2 The YAS software language repository

Artifact in a paper

2.1 Examples of languages

Figure 2 shows basic representation types in YAS and a few more specific software languages related to different aspects of a simple language BNL—*B*inary *N*umber *L*anguage. The nodes in the figure denote languages including 'formats' (e.g., XML-based ones) or general 'representation types' (e.g., *text*). The directed edges (arrows) denote subset relationship for languages in a set-theoretical sense. For instance, language *bnl*(*text*) corresponds to the concrete textual syntax of BNL. Thus, language *text* can be viewed as the universe for text-based languages. We explain the various languages in the sequel.

Here is an example of a binary number represented as text, i.e., an element of *bnl*(*text*):

Text resource *languages/BNL/samples/5comma25.bnl*

```
101.01
```

Language *bnl*(*json*) corresponds to the abstract, tree-based syntax of BNL using JSON for representation; here is the JSON representation of '101.01':

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Figure 2 shows basic representation types in YAS and a few more
languages related to different aspects of a simple language BNL–
*L*anguage. The nodes in the figure denote languages including 'for
based ones) or general 'representation types' (e.g., *text*). The directe
denote subset relationship for languages in a set-theoretical sen
language *bnl*(*text*) corresponds to the concrete textual syntax of BNL
text can be viewed as the universe for text-based languages. We ex
languages in the sequel.

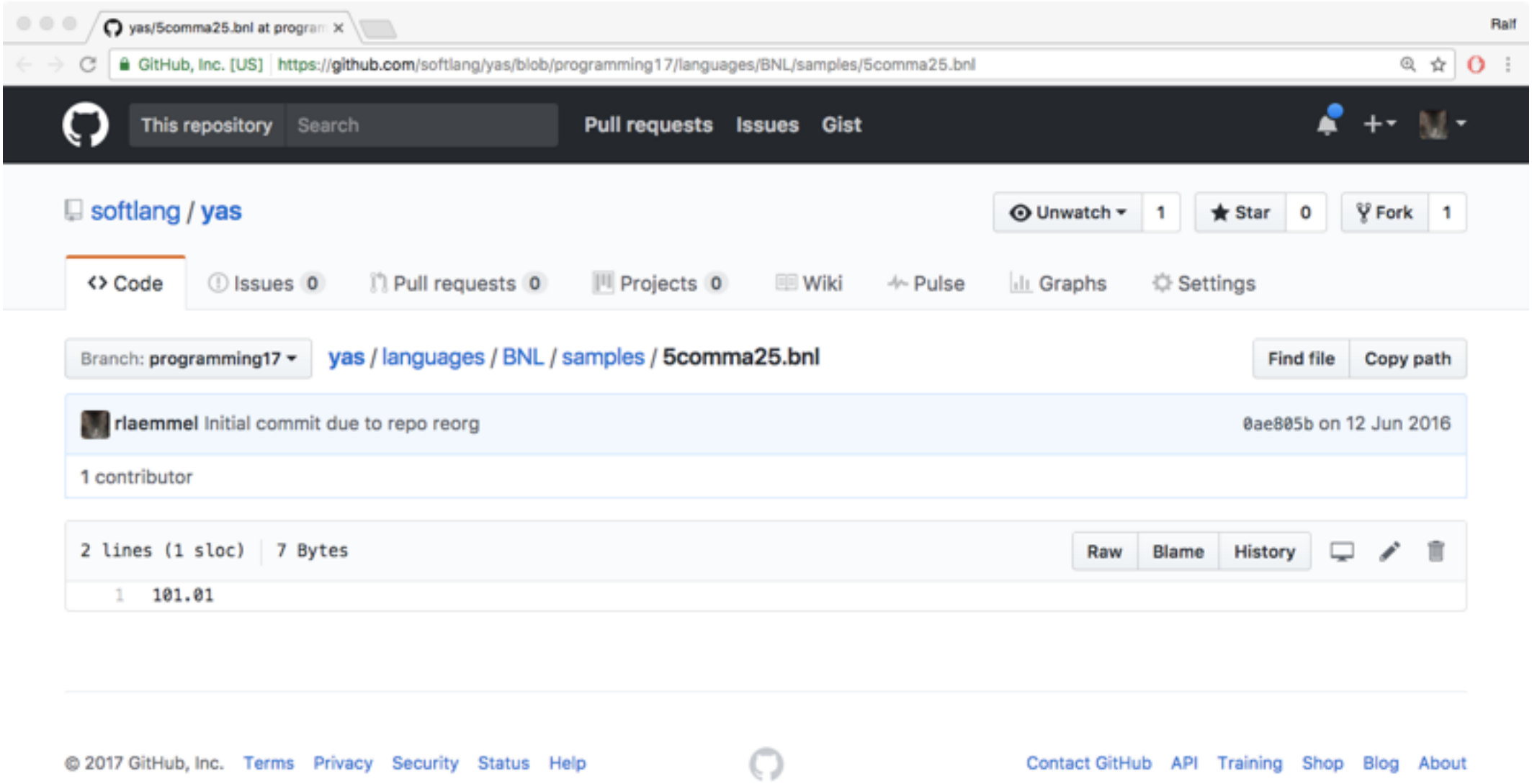
Here is an example of a binary number represented as text, i.e
bnl(*text*):

Text resource *languages/BNL/samples/5comma25.bnl*

```
101.01
```

Language *bnl*(*json*) corresponds to the abstract, tree-based synt
JSON for representation; here is the JSON representation of '101.01

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Branch: **programming17** ▾ yas / languages / BNL / samples / **5comma25.bnl**

rlaemmel Initial commit due to repo reorg

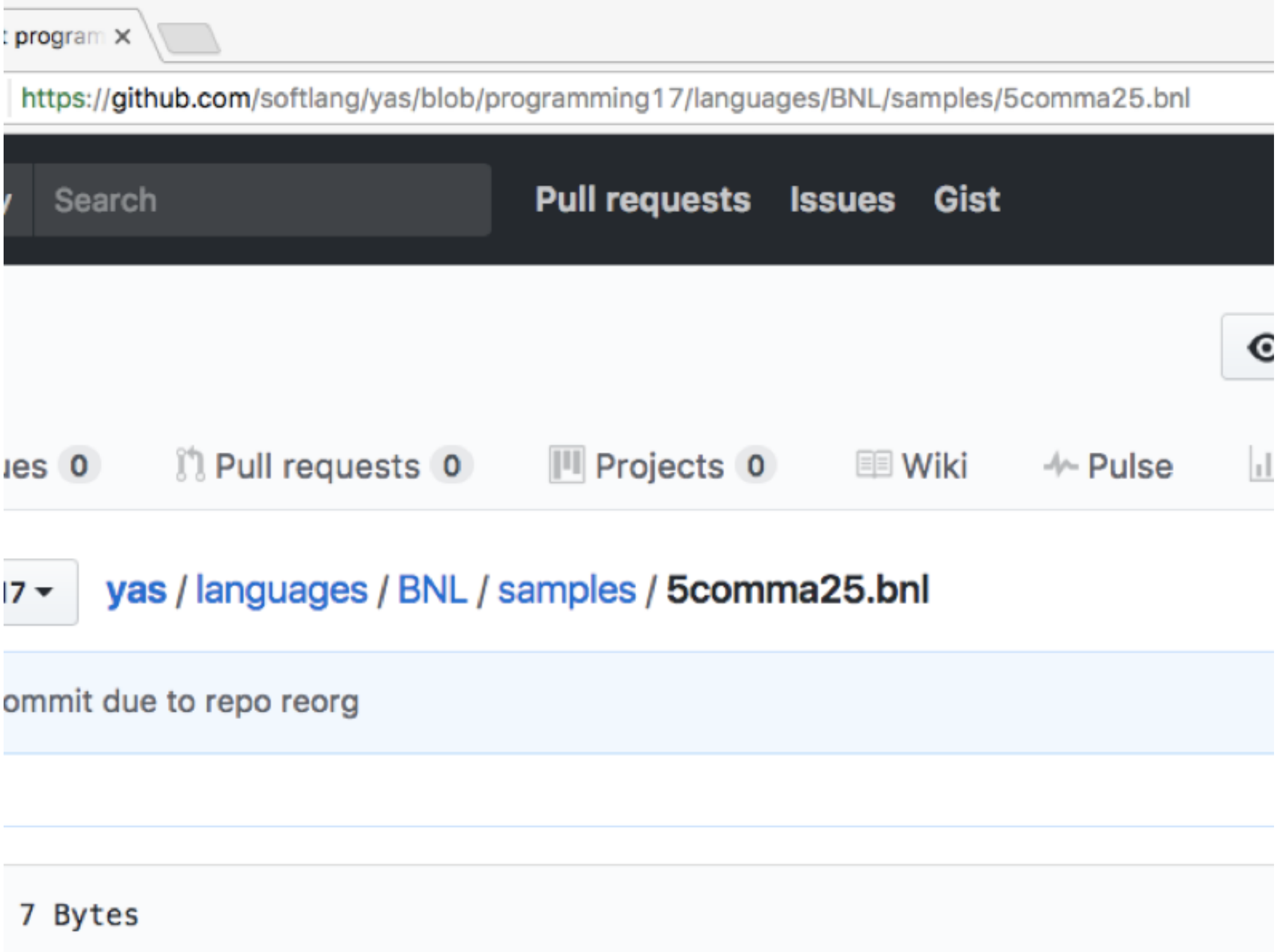
1 contributor

2 lines (1 sloc) | 7 Bytes

```
1  101.01
```

© 2017 GitHub, Inc. Terms Privacy Security Status Help

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.


https://github.com/softlang/yas/blob/programming17/languages/BNL/samples/5comma25.bnl
Search
Pull requests
Issues
Gist
Pull requests 0
Projects 0
Wiki
Pulse
yas / languages / BNL / samples / 5comma25.bnl
ommit due to repo reorg
7 Bytes

Decimal representation of *101.01*

5.25.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A JSON-based AST of *101.01*

```
{
   "bits": ["one", "zero", "one"],
   "rest": ["zero", "one"]
}
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Symbolic conversion from 101.01 to 5.25

```
2^ (1+1+1—1)+ (0+2^ (1+1+1—1—1—1))+ (0+2^ (—1—1)).
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An ANTLR grammar for binary numbers

```
grammar BnlEbnf;
@header { package org.softlang; }
number  : bit+ ('.' bit+)? WS? EOF;
bit  : '0' | '1';
WS : [ \t\n\r]+ ;
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A BNF for binary numbers

```
[number] number : bits rest ;
[single] bits : bit ;
[many] bits : bit bits ;
[zero] bit : '0' ;
[one] bit : '1' ;
[integer] rest : ;
[rational] rest : '.' bits ;
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An algebraic signature for binary numbers

```
symbol number: bits × rest → number ;
symbol single: bit → bits ;
symbol many: bit × bits → bits ;
symbol zero: → bit ;
symbol one: → bit ;
symbol integer: → rest ;
symbol rational: bits → rest ;
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A DCG for binary-to-decimal number of conversion

101.01

2^ (1+1+1—1)+ (0+2^ (1+1+1—1—1—1))+ (0+2^ (—1—1)).

```
number(Val1+Val2) ⟶ bits(Len1—1, Len1, Val1), rest(Val2).
bits(Pos, 1, Val) ⟶ bit(Pos, Val).
bits(Pos0, Len1+1, Val1+Val2) ⟶ bit(Pos0, Val1), bits(Pos0—1, Len1, Val2).
bit(_Pos, 0) ⟶ ['0'].
bit(Pos, 2^Pos) ⟶ ['1'].
rest(0) ⟶ [].
rest(Val) ⟶ ['.'], bits(—1, _Len, Val).
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

An evaluator

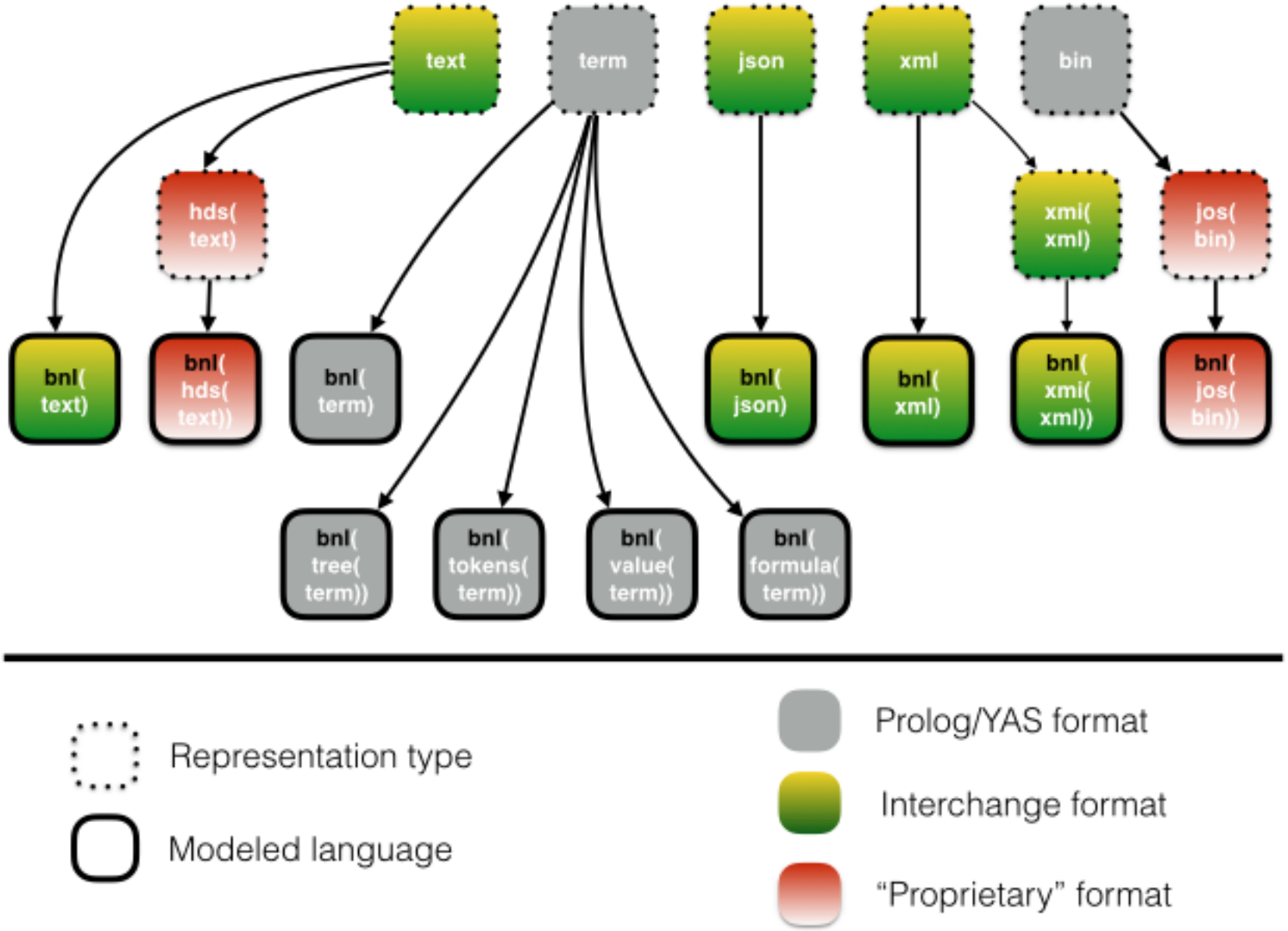
```
2^ (1+1+1—1)+ (0+2^ (1+1+1—1—1—1))+ (0+2^ (—1—1)).
```

```
evaluate(F, V) ⇐ V is F.
```

```
5.25.
```

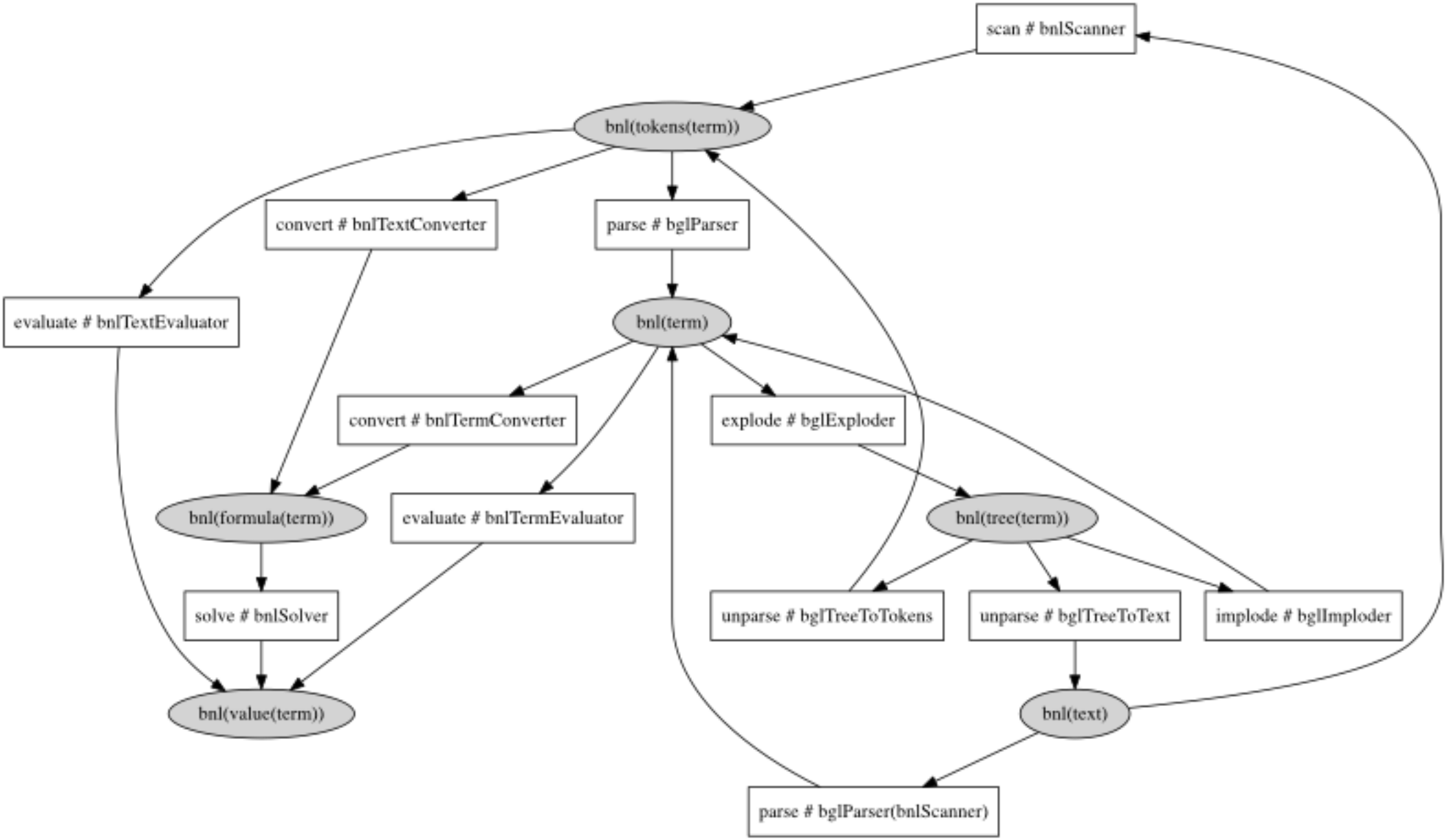
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Languages as types in an SLR



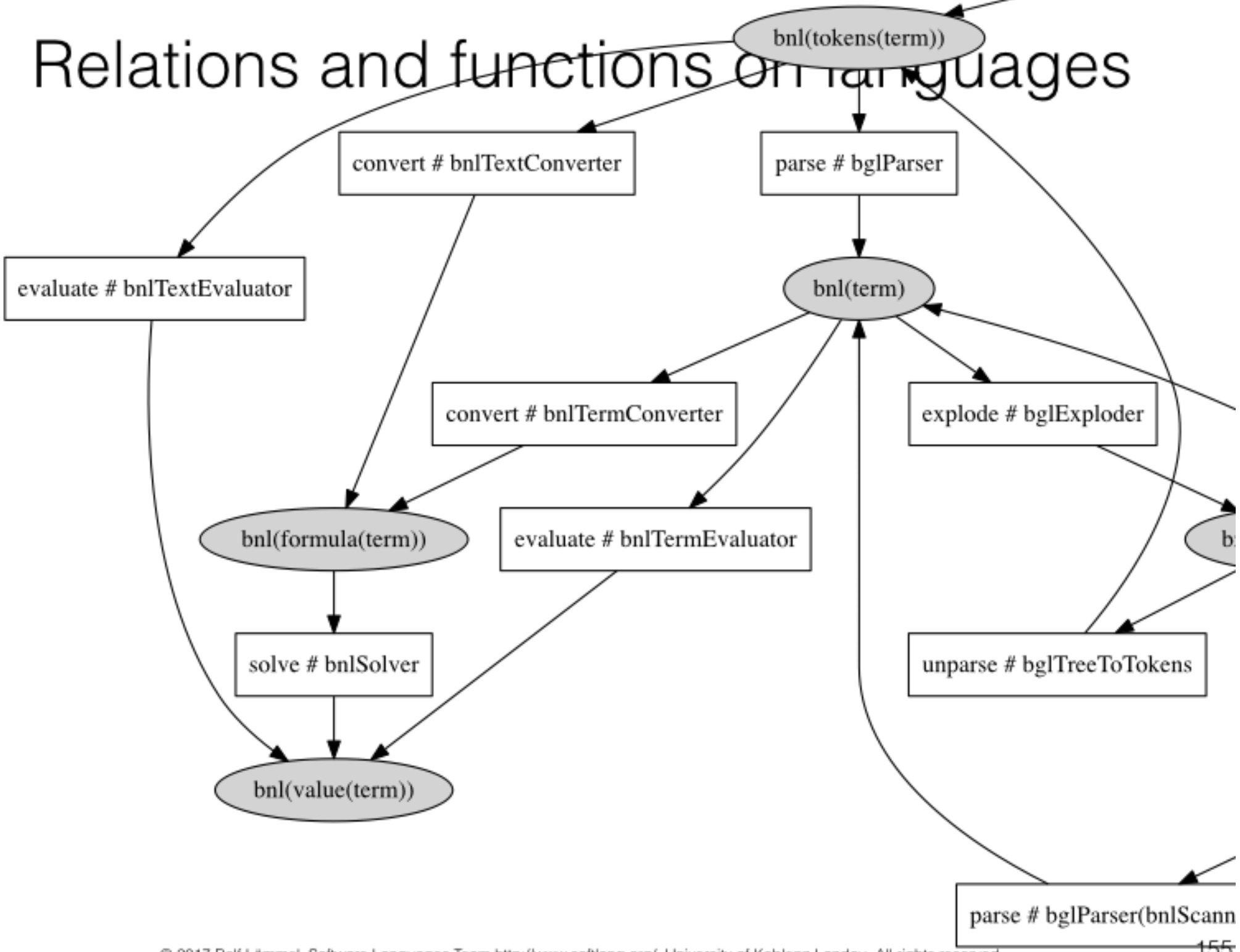
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Relations and functions on languages



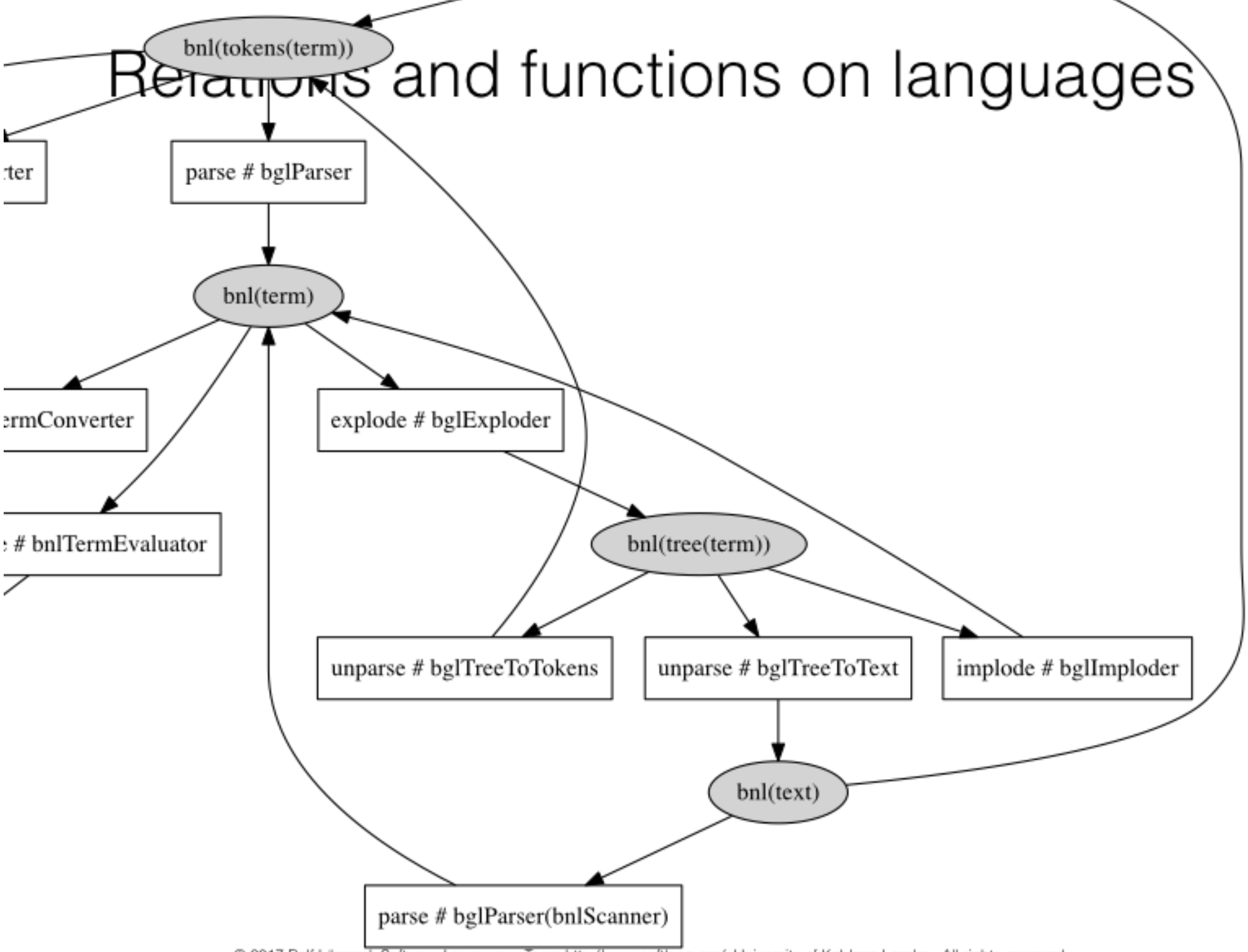
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Relations and functions on languages



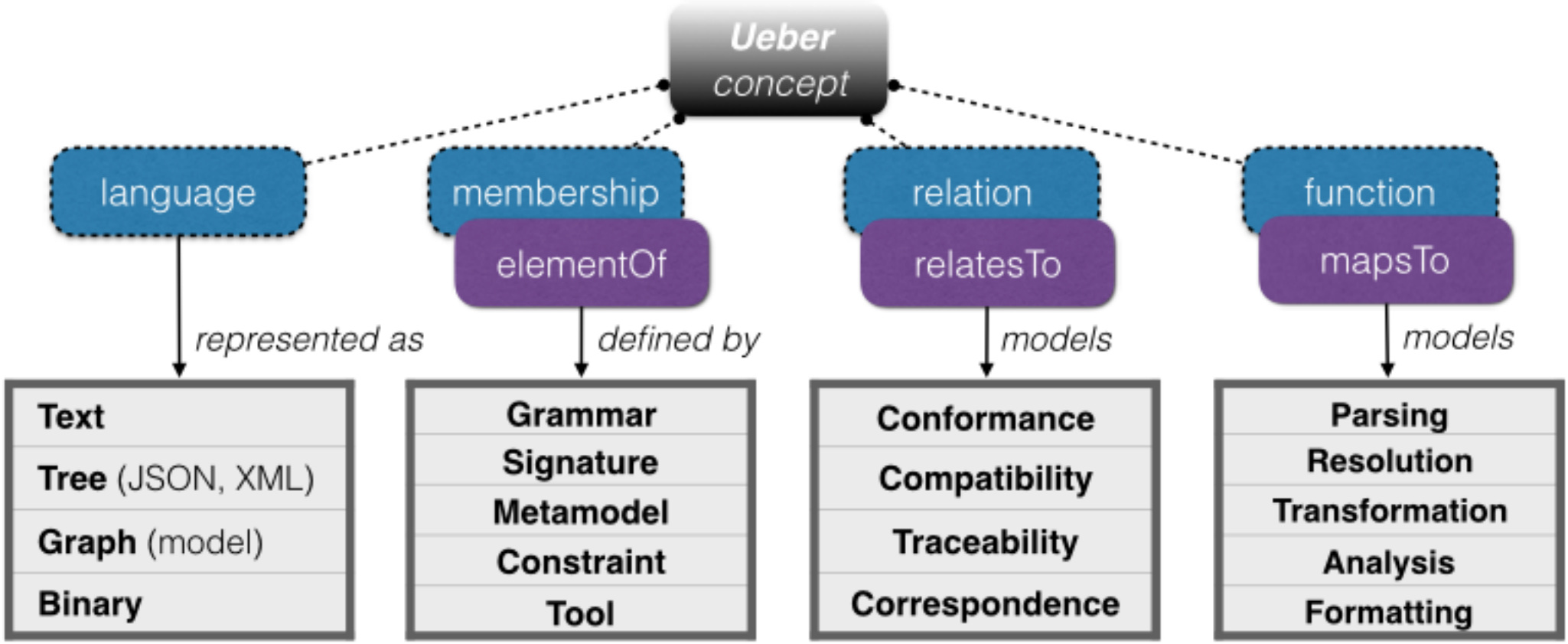
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Relations and functions on languages



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

ueber's language concepts



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

A small megamodel

This is about text and term representation of binary numbers including conformance and parsing.

language(text).

language(term).

language(bnl(text)).

language(bnl(term)).

elementOf('…/samples/5comma25.bnl', bnl(text)).

elementOf('…/samples/5comma25.term', bnl(term)).

membership(bnl(text), bglAcceptor(bnlScanner), ['languages/BNL/cs.term']).

membership(bnl(term), bslAcceptor, ['languages/BNL/as.term']).

relation(conformsTo, [term, bsl(term)], bslConformance, []).

relatesTo(conformsTo, ['…/samples/5comma25.term', 'languages/BNL/as.term']).

function(parse, [bnl(text)], [bnl(term)], bglParser(bnlScanner), ['languages/BNL/cs.term']).

mapsTo(parse, ['…/samples/5comma25.bnl'], ['…/samples/5comma25.term']).

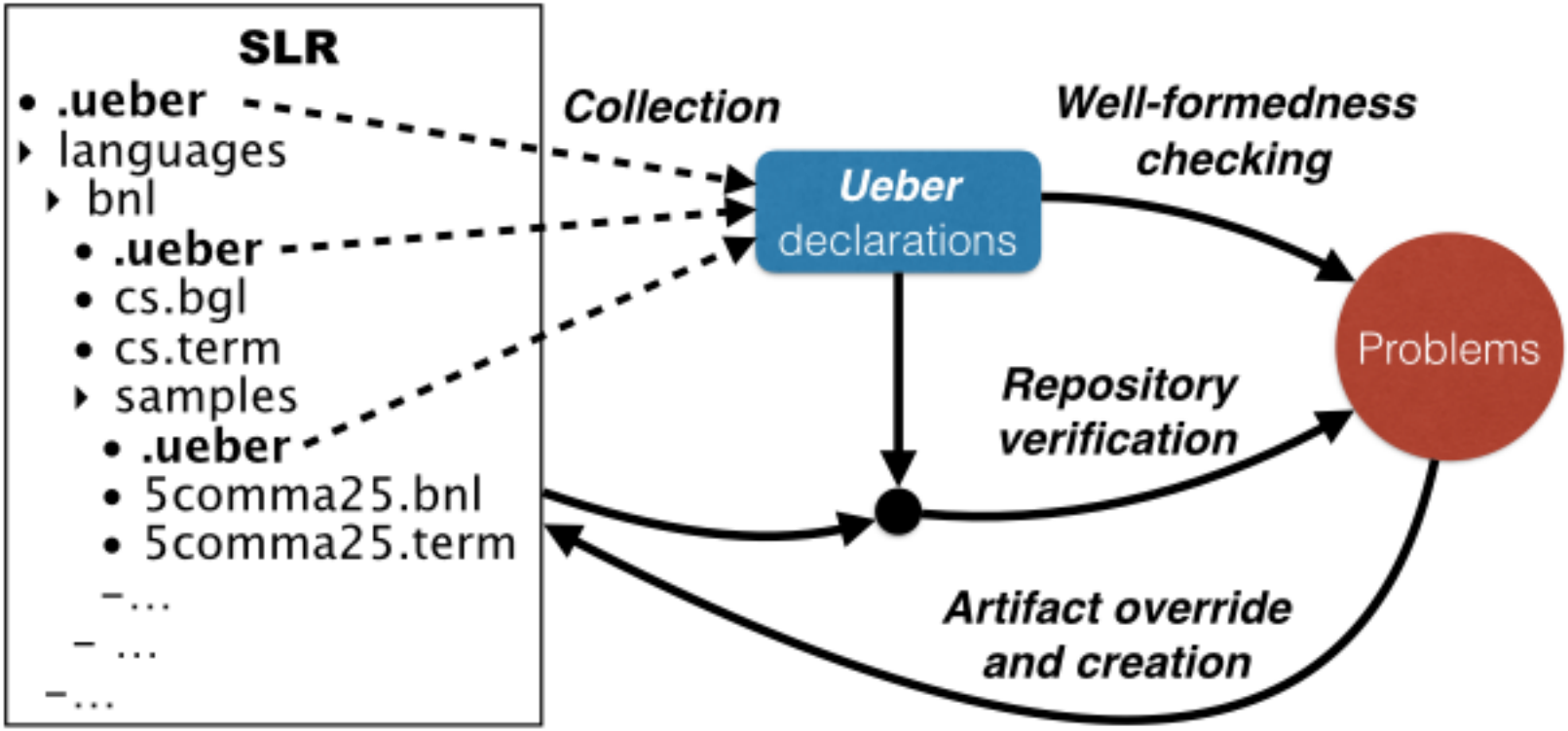
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Abstract syntax of *ueber*

```
type model = decl* ;
symbol language : lang → decl ;
symbol elementOf : file × lang → decl ;
symbol notElementOf : file × lang → decl ;
symbol membership : lang × goal × file* → decl ;
symbol relation : rela × lang* × goal × file* → decl ;
symbol relatesTo : rela × file* → decl ;
symbol function : func × lang* × lang* × goal × file* → decl ;
symbol mapsTo : func × file* × file* → decl ;
symbol equivalence : lang × goal × file* → decl ;
symbol normalization : lang × goal × file* → decl ;
symbol macro : goal → decl ;
type file = string ; // filenames
type rela = string ; // names of relations
type func = string ; // names of functions
type lang = term ; // names of languages
type goal = term ; // Prolog literals
```

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

ueber's semantics



© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Output produced by the *ueber* processor

'Breaking changes to a language processor'

- Baseline *languages/PPL/tests/hseplist.txt*: **disagreeing**.
- mapsTo(pp,[*languages/PPL/tests/hseplist.ppl*],[*....txt*]): **UNVERIFIED**.

'Development of a new test case'

- Baseline *languages/PPL/tests/indent.txt*: **missing**.
- elementOf(*languages/PPL/tests/indent.txt*,text): **UNVERIFIED**.
- mapsTo(pp,[*languages/PPL/tests/indent.ppl*],[*....txt*]): **UNVERIFIED**.

'Modeling a new relationship'

- Overload evaluate: ([*languages/BNL/samples/5comma25.bnl*]) -> ([*....value*]): **missing**.
- mapsTo(evaluate,[*languages/BNL/samples/5comma25.bnl*],[*....value*]): **NOT OK**.
- mapsTo(evaluate,[*languages/BNL/samples/5comma25.bnl*],[*....value*]): **UNVERIFIED**.

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Integrated compile- and run-time

	Prolog	Haskell	Java	Python
Implement relations & functions	Predicates	Main functions	Main methods	Scripts
Code location	Module auto loading	Automated module search path	Automated CLASSPATH	Automated PYTHONPATH
Compilation	N/A	On the fly	On the fly	N/A

Representation across different implementation languages

	Prolog	Haskell	Java	Python
Parse text	DCG, …	Parsec, …	ANTLR, …	ANTLR, …
Represent trees	Prolog terms JSON, XML, …	read/show conversion, JSON, XML	JSON, XML	str/repr conversion, JSON, XML
Represent graphs	Prolog terms	N/A	Serializable objects, XMI, …	Serializable objects, …

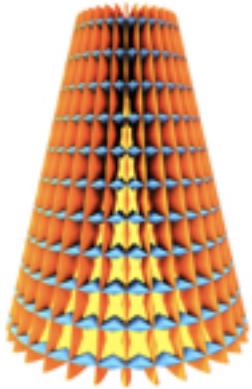
© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

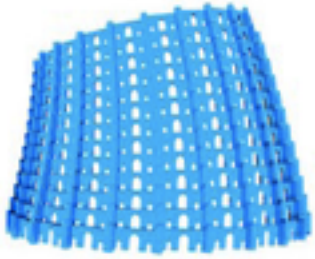
Advanced aspect of the YAS SLR

- Semantic annotations and LOD
- Incremental building
- Test-data generation
- Migration of a project to *ueber*
- Integration with version control
- Infer *ueber* declarations based on conventions
- A useful model of language variants

© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.

Thank you!





© 2017 Ralf Lämmel, Software Languages Team http://www.softlang.org/, University of Koblenz-Landau. All rights reserved.