Formalizing Language Definitions
using ASF+SDF

Formal Methods Course

Utrecht, April 16, 1997

Arie van Deursen
CWI

Eelco Visser
University of Amsterdam
Schedule

• 09:30 – 10:00; Introduction (Arie)

• 10:00 – 11:00; Exercises

• 11:00 – 11:30;
  Foundations / Details (Arie)

• 11:30 – 12:30; Formalizing programming language concepts (Eelco)

• 12:30 – 13:30; Lunch

• 13:30 – 15:30; Exercises:
  programming language concepts

• 15:30 – 16:30; Applications, research, future developments (Arie)
Aims and Scope

Language definitions:

- Understandable:
  ⇒ Easy to reason about;

- Executable
  ⇒ Experimental language prototyping;

- Readable
  ⇒ Literate specification.
Language Definitions

- Lexical, context-free, abstract syntax;
  ⇒ Scanner, parser, syntax-directed editor.

- Context-sensitive requirements
  ⇒ type checker.

- Evaluation (operational semantics)
  ⇒ interpreter.

- Mapping to other language
  ⇒ compiler.

- Transformations
  ⇒ optimizer, restructuring tool

- ...

Utrecht, April 1997
ASF+SDF

- SDF: Syntax Definition Formalism
- ASF: Algebraic Specification Formalism
- ASF+SDF Meta-environment:
  - Executing ASF+SDF specifications
  - Developing ASF+SDF specifications
- Environment for generating environments.
Booleans in ASF+SDF

imports Layout
exports
sorts BOOL
context-free syntax
    true          -> BOOL
    false         -> BOOL
    BOOL and BOOL -> BOOL {left}
    BOOL or BOOL  -> BOOL {left}
    not BOOL      -> BOOL
    "(" BOOL ")"  -> BOOL {bracket}
priorities
    not > and > or
variables
    P -> BOOL
Booleans in ASF+SDF (cont.)

equations

[o1] true or P = true
[o2] false or P = P
[a1] true and P = P
[a2] false and P = false
[n1] not true = false
[n2] not false = true
Signatures, Grammars

• SDF: “BOOL and BOOL \( \rightarrow \) BOOL”:
  
  – Declares a function in a signature:
    and: \( BOOL \times BOOL \rightarrow BOOL \)
  
  – Declares a production in a grammar:
    \( \langle BOOL \rangle ::= \langle BOOL \rangle ”and” \langle BOOL \rangle \)

• Sentence ”true and false” corresponds to term and(true,false).

• Parser: map sentence to term.

• Functions: computed by term rewriting.
Lambda-Calculus

imports Layout
exports
  sorts ID L-EXP
  lexical syntax
    [a-z][a-z0-9]*   -> ID
  context-free syntax
    ID               -> L-EXP
    lambda ID "."   L-EXP   -> L-EXP
    L-EXP L-EXP     -> L-EXP {left}
    "(" L-EXP ")"   -> L-EXP {bracket}
priorities
  { L-EXP L-EXP   -> L-EXP } > { lambda "." } 
variables
  E[0-9’]*        -> L-EXP
  V[0-9’]*        -> ID
Lists

imports Lambda-Calculus Booleans
exports
  sorts ID-LIST
  context-free syntax
    "[" {ID ","}* "]" -> ID-LIST
    ID member-of ID-LIST -> BOOL
    ID-LIST "++" ID-LIST -> ID-LIST {left}
    ID-LIST "-" ID -> ID-LIST
variables
  Id [0-9]* -> ID
  Id [0-9]* "*" -> ID*
  L [0-9]* -> ID-LIST
Lists (II)

equations

[c1]  \[\text{Id}^\ast \] \text{++} \ [\text{Id'}^\ast] = \text{[Id}^\ast, \text{Id'}^\ast]\]

[m1]  \text{Id member-of [] = false}

[m2]  \text{Id member-of [Id, Id}^\ast] = \text{true}

[m3]  \text{Id} != \text{Id'}

=================================================================
\text{Id member-of [Id', Id}^\ast] = \text{Id member-of [Id]}

[m1]  \text{Id member-of L = false}

=================================================================
\text{L - Id} = \text{L}

[m2]  \text{[Id}^\ast, \text{Id, Id'}^\ast] - \text{Id} = \text{[Id}^\ast, \text{Id'}^\ast] - \text{Id}

---

\textit{Utrecht, April 1997}
Free variables

imports Id-Lists
exports
  context-free syntax
    "FV" ( L-EXP )  -> ID-LIST

equations

  [f1]  FV( V ) = [ V ]

  [f2]  FV( E1 E2 ) = FV(E1) ++ FV(E2)

  [f3]  FV( lambda V . E ) = FV(E) - V
Exercises (1 hour)

• ASF+SDF User’s Manual: Guided Tour:
  Definition of syntax, type checking, and evaluation of “Pico”.

• Questions on ASF+SDF formalism:
  – User’s Manual;
  – Chapter 1 of “Language Prototyping”;
  – Ask us!
Models

- Algebra $\langle A, O \rangle$;
  
  $A$: set of values
  $O$: set of operations $A \rightarrow A$.

- An algebra is a model of a spec by:
  
  - mapping functions in the signature to operations.
  
  - such that equations are satisfied.

- Equational calculus for terms ("replacing equals by equals"): 
  
  Gives equalities valid in all models.
Initial Models

- Initial model: No junk, no confusion.
  - Booleans with junk: extra value for “unknown”.
  - Booleans with confusion: true = false.

- Values: ground constructor terms;

- Operators: defined functions;

- Structural induction for open equalities.
Term Rewriting

- Confluence – reduction order irrelevant;

- Termination (strong normalization);

- Constructors: symbols used for building normal forms.

- Sufficient completeness:
  Defined functions cover all cases.

- Termination / confluence / completeness:
  show by constructor case distinction.
Conditional Rewriting

- Positive / negative rewriting.
- Normalize condition sides; compare normal forms.
- Confluence and termination required.
- Condition sides can introduce new variables: effect of “let-construct”.

\[
tc(E1, Env) = \langle \text{Type1}, \text{Err1} \rangle,
\]
\[
tc(E2, Env) = \langle \text{Type2}, \text{Err2} \rangle
\]
\[
\text{compatible}(\text{Type1}, \text{Type2}) = \text{true}
\]
\[
\text{tc}(E1 \ 0p \ E2, Env) = \langle \text{Type1}, \text{Err1} + \text{Err2} \rangle
\]
Default Equations

• Otherwise / default equations can be used to “cover remaining cases”.

  • \([t1]\) \(\text{is-plus-op}( E1 + E2 ) = \text{true} \)
    \([\text{default-t2}] \) \(\text{is-plus-op}( E ) = \text{false} \)

• Abbreviation for explicit equations covering remaining constructors.

• Operationally: applied last.
Associative Lists

- Rewriting: repeatedly search redex: match; instantiate; validate conds; replace.

- Associative matching: not unique.
  
  pattern: \([X^*, X'^*]\); term: \([1]\);
  match 1: \(X^* = 1, X'^* = \text{empty}\);
  match 2: \(X^* = \text{empty}, X'^* = 1\).

- Associative ⊢ conditional rewriting:
  Backtracking required to find list-match that satisfies the conditions.

- General AC matching − too expensive; not implemented.
Injections

• ASF+SDF permits “syntax-less” chain rules for injecting NAT into EXP.

```
context-free syntax
    NAT  ->  EXP
```

• If NAT is injected via more than one route into EXP, ambiguities arise.
Lexical syntax

- Lexical analysis: map stream of characters to sequence of tokens.

- Lexical definitions:
  - Character class: \([A-Z0-9]+\) \(\rightarrow\) ID
  - Negated class: "\\" ~["\"]* "\\" \(\rightarrow\) STRING

- Lexical sort SORT induces function:
  sort( CHAR+ ) \(\rightarrow\) SORT
Modularization

• Exported / hidden items;

• Import of (exported items) from modules.

• Desirable, but absent features:
  – renaming of sorts or functions
  – parameterized modules

Now: use work-around.
Future: ASF+SDF+Renamings.
Literate specification

- Executable specification ≡
  Readable definition

- Generate $\text{\LaTeX}$ from ASF+SDF

- Extensible list of mappings between
  ASCII tokens ("++", ">=", \(\rightarrow\))
  and $\text{\LaTeX}$ symbols (\(\leftrightarrow\), \(\geq\), \(\rightarrow\)).

- Comment directly passed to $\text{\LaTeX}$.

- Use it for writing papers!