Executable Language Definitions in ASF+SDF

Arie van Deursen
CWI, Amsterdam

Note: Supported by ESPRIT G.I.P.E.-II, and NWO project *Incremental Program Generators*

Executable Language Definitions

Case Studies and

Origin Tracking Techniques

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam,
op gezag van de Rector Magnificus
Prof.dr P.W.M. de Meijer
ten overstaan van
een door het college van dekanen ingestelde
commissie
in het openbaar te verdedigen
in de Aula der Universiteit
(Oude Lutherse Kerk, ingang Singel 411, hoek Spui)
op donderdag 29 september 1994 te 13:30 uur

door

Arie van Deursen

geboren te 's Gravenhage.

Language Design

- New languages pop up everywhere: programming, specifying, databases, ...

- **Formal definitions**:
  - Help to describe and analyze a language;
  - Covering, e.g., syntax, static analysis, semantics (operational, denotational, etc.), transformations, translations, ...

- **Tools** are needed for:
  - Parsing, type checking, transforming, translating, evaluating, debugging, ...
  - Conducting experiments during language design.

Tool Generators

- Support during language design:
  - **Specify** language;
  - **Generate** tools immediately;
- Well-known examples:
  - Based on attribute grammars, operational or denotational semantics, ...
  - Synthesizer Generator, Programming System Generator PSG, Gandalf, the Pan language-based system, Eli, ...
- In this talk: ASF+SDF

ASF+SDF

- The ASF+SDF formalism:
  - Algebraic Specification Formalism; Syntax Definition Formalism.
  - First-order conditional equations, modular, user-definable syntax.

- The ASF+SDF **Meta-environment**:
  - Generates **parsers** and **editors**;
  - Executes ASF+SDF using **rewriting**;
  - Generate language-specific environments incrementally.

A Grammar in ASF+SDF

```
%% module Small-Language

imports Integers Identifiers

exports
  sorts
    STAT EXP
  context-free syntax
    INT                          -> EXP
    ID ":=" EXP                  -> STAT
    "if" EXP "then" STAT
     "else" STAT "fi"            -> STAT
```

Assembly code in ASF+SDF (1)

```
%% module Assembly
imports Integers Identifiers
exports
  sorts INSTR CODE LABEL
  lexical syntax
    [0-9]+                -> LABEL
  context-free syntax
    "cjump" LABEL         -> INSTR
    "jump" LABEL          -> INSTR
    "lab" LABEL           -> INSTR
    "push" INT            -> INSTR
    "push" ID             -> INSTR
    "lvar"                -> INSTR
    "rvar"                -> INSTR
    "move"                -> INSTR
    INSTR*                -> CODE
    CODE "++" CODE        -> CODE {left}
    LABEL LABEL           -> LABEL
```

A Compilation (1)

```
%% module Compile

imports Small-Language Assembly

exports
  context-free syntax
    cmp-stat( STAT, LABEL )  -> CODE
    cmp-exp( EXP )           -> CODE

hiddens
  variables
    S['']* -> STAT
    E      -> EXP
    L      -> LABEL
    Int    -> INT
    Id     -> ID
```

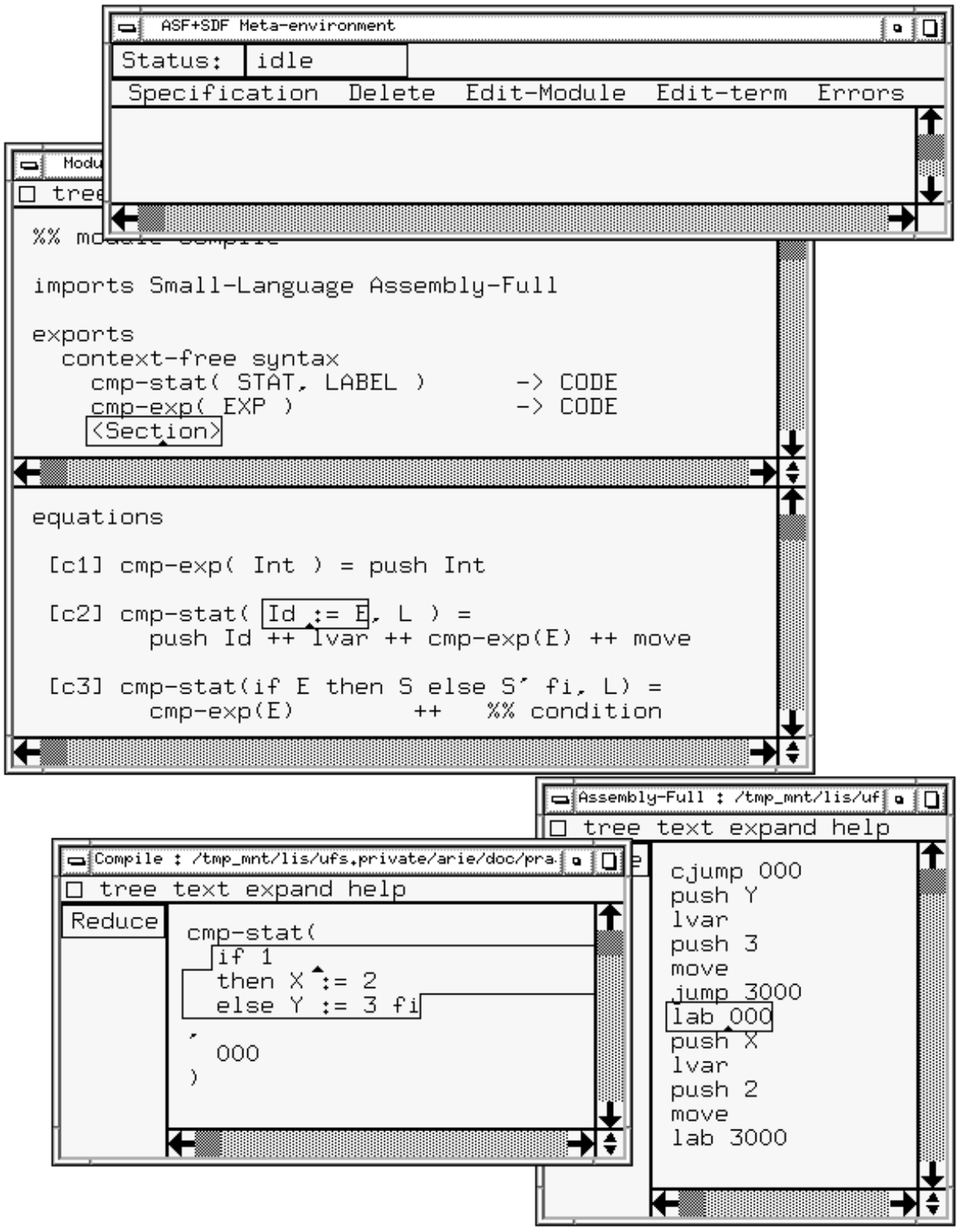
A Compilation (2)

```
equations

 [c1] cmp-exp( Int ) = push Int

 [c2] cmp-stat( Id := E, L ) =
       push Id ++ lvar ++ cmp-exp(E) ++ move

 [c3] cmp-stat(if E then S else S' fi, L) =
       cmp-exp(E)          ++  %% condition
       cjump L             ++
       cmp-stat(S', 1L)    ++  %% else part
       jump 3L             ++
       lab L               ++  %% then part
       cmp-stat(S, 2L)     ++
       lab 3L
```


ASF+SDF Meta-environment
Status: idle
Specification Delete Edit-Module Edit-term Errors
tree
%% module compile
imports Small-Language Assembly-Full
exports
context-free syntax
cmp-stat(STAT, LABEL) -> CODE
cmp-exp(EXP) -> CODE
<Section>
equations
[c1] cmp-exp(Int) = push Int
[c2] cmp-stat(Id := E, L) =
push Id ++ lvar ++ cmp-exp(E) ++ move
[c3] cmp-stat(if E then S else S' fi, L) =
cmp-exp(E) ++ %% condition
Assembly-Full : /tmp_mnt/lis/uf
tree text expand help
cjump 000
push Y
lvar
push 3
move
jump 3000
lab 000
push X
lvar
push 2
move
lab 3000
Compile : /tmp_mnt/lis/ufs.private/arie/doc/pra
tree text expand help
Reduce
cmp-stat(
if 1
then X := 2
else Y := 3 fi
,
000
)

Execution Using Term Rewriting

- Specifications are executed using **conditional term rewriting**.

- Some examples:
  - Translators
    mapping source to target language;
  - Type checkers
    mapping language to error messages;
  - Evaluators
    computing changes in store.

- Is rewriting all we want?

Origin Tracking

- **Rewriting**: just compute a **result**.

- Rewriting plus **origin tracking**:
  maintain a relation between the initial term and the result term as well.

- Used to obtain from same the definitions:

  - Animator
    Highlight statement executed;

  - Error handler
    Display *where* error occurred;

  - Source-level debugger
    Help to debug a compiled program.

The Origin Function

For every elementary reduction step:

- $t \equiv C[\alpha^\sigma] \to C[\beta^\sigma] \equiv t'$:

  Apply rewrite rule $\alpha \to \beta$,
  under substitution σ,
  in term t with redex at *occurrence* u,
  resulting in t'.

  An **occurrence** identifies a subterm.

- Associate with it a function
  $org : \mathcal{O}(t') \to \mathcal{P}(\mathcal{O}(t))$
  mapping occurrences in t' to sets of occurrences in t'.

Distinguish: *context* C, *variables* in α and β, and *function-symbols* in α and β.

The Origin Function (2)

equations

```
 [c1] cmp-exp( Int ) = push Int

 [c2] cmp-stat( Id := E, L ) =
       push Id ++ lvar ++ cmp-exp(E) ++ move

 [c3] cmp-stat(if E then S else S' fi, L) =
       cmp-exp(E)          ++  %% condition
       cjump L             ++
       cmp-stat(S', 1L)    ++  %% else part
       jump 3L             ++
       lab L               ++  %% then part
       cmp-stat(S, 2L)     ++
       lab 3L
```

Primitive Recursive Schemes

Algebraic specification $\langle \Sigma, E \rangle$
$\Sigma = G \cup S \cup \Phi, \qquad E = E_\Phi \cup E_S.$

- G: signature for program constructors;
  $g : G_1 \times \cdots \times G_n \to G_0$

- Φ: recursively defined functions;
  $\phi : G_0 \times S_1 \times \cdots \times S_m \to S_0$

- S: auxiliary functions, defined by E_S.

- E_Φ is the set of *recursion equations*:
  $\phi(p(x_1, \ldots, x_n), y_1, \ldots, y_m) = \tau$

Requirements: recursion equations should be strictly decreasing in G, and left-linear.

PRS Origins (2)

```
equations

 [c1] cmp-exp( Int ) = push Int

 [c2] cmp-stat( Id := E, L ) =
       push Id ++ lvar ++ cmp-exp(E) ++ move

 [c3] cmp-stat(if E then S else S' fi, L) =
       cmp-exp(E)          ++  %% condition
       cjump L             ++
       cmp-stat(S', 1L)    ++  %% else part
       jump 3L             ++
       lab L               ++  %% then part
       cmp-stat(S, 2L)     ++
       lab 3L
```

Origin Tracking

- Based on theoretical work on **residual maps** by, e.g., Huet and Lévy;
- Notion of **origin** and its applications due to Bertot (ESOP'90, CAAP'92);
- Conditional, non-orthogonal TRSs: 1993, J.Sym.Comp., joint work with Klint, Tip.
- Extension to Higher-order specifications; joint work with Dinesh; HOA'93.
- Proposal for PRSs; CSN'93.
- Dependence Tracking; applications to Slicing; (Field and Tip; PLILP '94)

The ASF+SDF Project

ASF+SDF is used as a research platform:

- User definable syntax and incremental parser generation;
  (Heering, Klint, Rekers)

- Literate Specification Techniques
  (Klint, Visser)

- Incremental Rewriting
  (Van der Meulen)

- Higher-order algebraic specifications
  (Heering)

- User interfaces and visual languages
  (Koorn, Uskudarli)

Type Checking Revisited

- "Classical" type checking: mapping from abstract syntax to {true, false} or to a domain of error values;

- Abstract Interpretation:
  Map syntax to domain of **types**, and execute program in that domain.

  integer + integer $\rightarrow$ integer

- Irreducible expressions correspond to type conflicts:

  integer + string

- Use of origin tracking to yield useful messages.

Applications (selection)

- LOTOS (Dutch PTT);
- Static semantics of Pascal and Eiffel;
- Process specification in μCRL (RUU);
- ASF+SDF $\rightarrow$ C;
- Compiler Construction (IBM NY);
- Trade in tulip bulbs;
- Teaching.

Financial Engineering

- Bank MeesPierson (Rotterdam): offers various **financial products**

- Competitive market; rapid introduction of new products.

- Flexibility of bank's automated systems (financial administration, management information)?

- Describe products using application language RISLA; and *generate* software.

- Used ASF+SDF to specify underlying data types, grammar, type checking, and translation to COBOL.

Tools for the "MN" formalism

- Action Semantics: denotations of programs are described by **action combinators**;
- Action-semantic descriptions are written using the "MN"-formalism (MN = Meta Notation);
- Tools built using ASF+SDF;
- Complications: user-definable syntax, user-definable type constructors, reloading of generated ASF+SDF modules;
- Used to teach Action Semantics, and to detect errors MN specifications.

Concluding Remarks

- The ASF+SDF Meta-Environment, a language design and implementation workbench;

- Origin Tracking, a technique to understand relations between initial data and final results;

- Various areas of ongoing research;

- Applicability in practice
  (academia and industry).