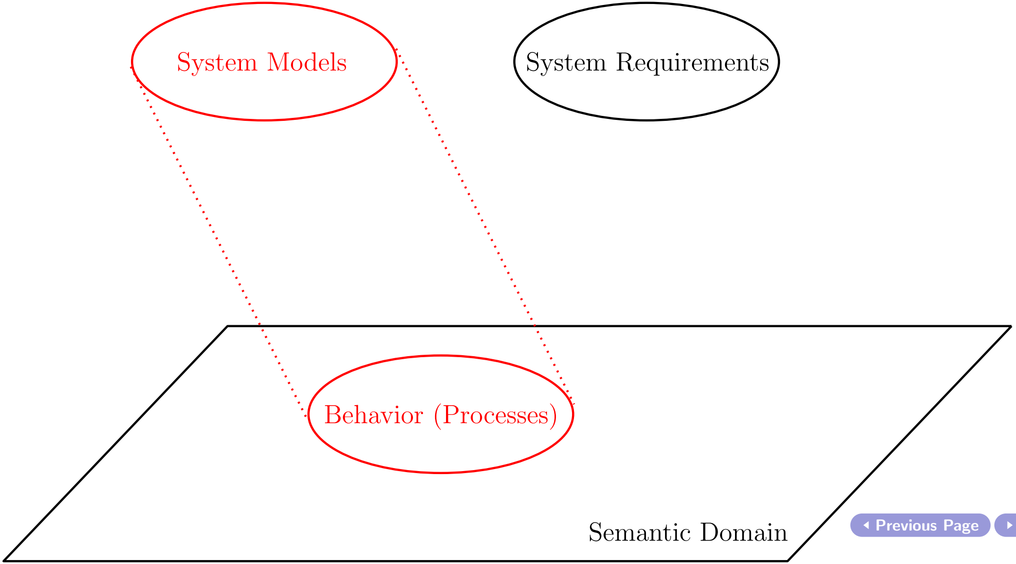
System Validation:
Defining Abstract Data Types

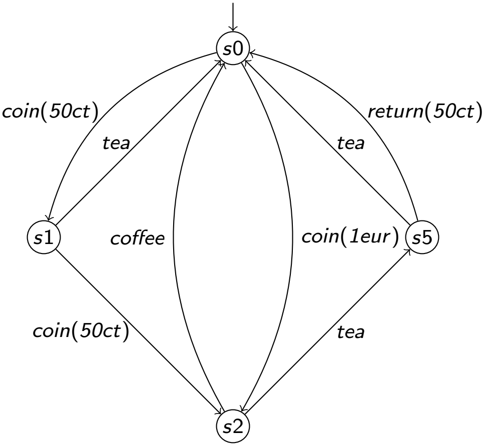
Mohammad Mousavi and Jeroen Keiren

General Overview



Motivating Example

Advanced Coffee Machine



Generic Concepts

Data types

- Classes: sorts

Generic Concepts

Data types

- Classes: sorts
- Elements: constructors

Data types

- Classes: sorts
- Elements: constructors
- Operations: maps
- Rules governing operations: equations

Example
Euro Sort

```
sort Euro;
```

Example
Euro Sort

```
sort Euro;
cons zero, fifty_cents,
     one_euro, more: Euro;
     % constants: constructors with no parameter
```

Previous Page Next Page

```
sort Euro;
cons zero, fifty_cents,
     one_euro, more: Euro;

map  eq: Euro × Euro → Bool;
     plus: Euro × Euro → Euro;
```

Example
Euro Sort

```
sort Euro;
cons zero, fifty_cents,
     one_euro, more: Euro;

map  eq: Euro × Euro → Bool;
     plus: Euro × Euro → Euro;
var  e:Euro;
eqn  eq(e, e)= true;                  (1)
     eq(zero, one_euro)= false;       (2)
     eq(one_euro, zero)= false;       (3)
     ...
```

Previous Page   Next Page

Example
Euro Sort (Cont'd)

`sort` Euro;

Example
Euro Sort (Cont'd)

```
sort Euro;
var  e: Euro;
eqn  plus(e,zero)= e;
     plus(zero,e)= e;
```

Example
Euro Sort (Cont'd)

```
sort Euro;
var  e: Euro;
eqn  plus(e,zero)= e;
     plus(zero,e)= e;
     plus(fifty_cents,fifty_cents)= one_euro;
```

Example
Natural

`sort` Natural;

Example
Natural

```
sort Natural;
cons zero: Natural;
     succ: Natural → Natural;
```

Example
Natural

```
sort Natural;
cons zero: Natural;
     succ: Natural → Natural;
map  eq: Natural × Natural → Bool;
```

```
sort Natural;
cons zero: Natural;
     succ: Natural → Natural;
map  eq: Natural × Natural → Bool;
var  i, j: Natural;
eqn  eq(i, i)= true;                   (1)
     eq(zero, succ(i))= false;         (2)
     eq(succ(i), zero)= false;         (3)
     eq(succ(i), succ(j))= eq(i,j);    (4)
```

Built-In Types

- Booleans: true, false, conjunction (&&), disjunction (||), negation (!), implication (=>), equality (==), quantifiers and much more.

Built-In Types

- Booleans: true, false, conjunction (&&), disjunction (||), negation (!), implication (=>), equality (==), quantifiers and much more.
- (Positive) Natural numbers: successor (succ), equality (==), maximum and minimum (max and min), addition (+), multiplication (*), division (div), modulo (mod) and much more.

Built-In Types

- Booleans: true, false, conjunction (&&), disjunction (||), negation (!), implication (=>), equality (==), quantifiers and much more.
- (Positive) Natural numbers: successor (succ), equality (==), maximum and minimum (max and min), addition (+), multiplication (*), division (div), modulo (mod) and much more.
- Integers: similar to above, predecessor (pred), minus (-), absolute (abs) and much more.

Built-In Types

- Booleans: true, false, conjunction (&&), disjunction (||), negation (!), implication (=>), equality (==), quantifiers and much more.
- (Positive) Natural numbers: successor (succ), equality (==), maximum and minimum (max and min), addition (+), multiplication (*), division (div), modulo (mod) and much more.
- Integers: similar to above, predecessor (pred), minus (-), absolute (abs) and much more.
- Reals

Built-In Types

- Booleans: true, false, conjunction (&&), disjunction (||), negation (!), implication (=>), equality (==), quantifiers and much more.
- (Positive) Natural numbers: successor (succ), equality (==), maximum and minimum (max and min), addition (+), multiplication (*), division (div), modulo (mod) and much more.
- Integers: similar to above, predecessor (pred), minus (-), absolute (abs) and much more.
- Reals
- Typecast: Pos2Nat, Nat2Pos, Int2Nat, etc.

Structured Types

- Syntax:

  sort St = struct elm_a | elm_b | f(s : S)

  ```
  sort  St
  cons  elm_a, elm_b: St;
        f: St → St;
  ```

Structured Types

- Syntax:
  sort St = struct elm_a?is_a | elm_b?is_b | f(s : S)?is_f
- Built-in recognizers
  `map` is_a, is_b, is_f: St $\rightarrow$ Bool;

Structured Types

- Syntax:
  sort St = struct elm_a?is_a | elm_b?is_b |
  f(s : S)?is_f
- Built-in recognizers
- Built-in equations for recognizers: provably different constructors

```
var   s :St;
eqn   is_a(elm_a)= true;
      is_a(elm_b)= false;
      is_a(f(s))= false;
      ...
```

Structured Types

- Syntax:
  sort St = struct elm_a?is_a | elm_b?is_b |
  f(s : S)?is_f
- Built-in recognizers
- Built-in equations for recognizers: provably different constructors
- Built-in equality, inequality and if-then-else maps

- Syntax: sort lst = List(St);

Constructed Types
Lists

- Syntax: sort lst = List(St);
- List enumeration: [elements] (comma separated)

Constructed Types
Lists

- Syntax: sort lst = List(St);
- List enumeration: [elements] (comma separated)
- Built-in equality and inequality, i-th element ($l.i$).

- Syntax: sort lst = List(St);
- List enumeration: [elements] (comma separated)
- Built-in equality and inequality, i-th element ($l.i$).
- Several built-in constructs and maps: cons ($|>$), concatenation ($++$), length ($\#$), member (in), head (head), tail (tail) and many more.

- Syntax: sort S = Set(St);

- Syntax: sort S = Set(St);
- Set enumeration: $\{a, b, \ldots\}$

- Syntax: sort S $=$ Bag(St)
- Set enumeration: $\{a, b, \ldots\}$
- Bag enumeration: $\{a : 3, b : 2, \ldots\}$

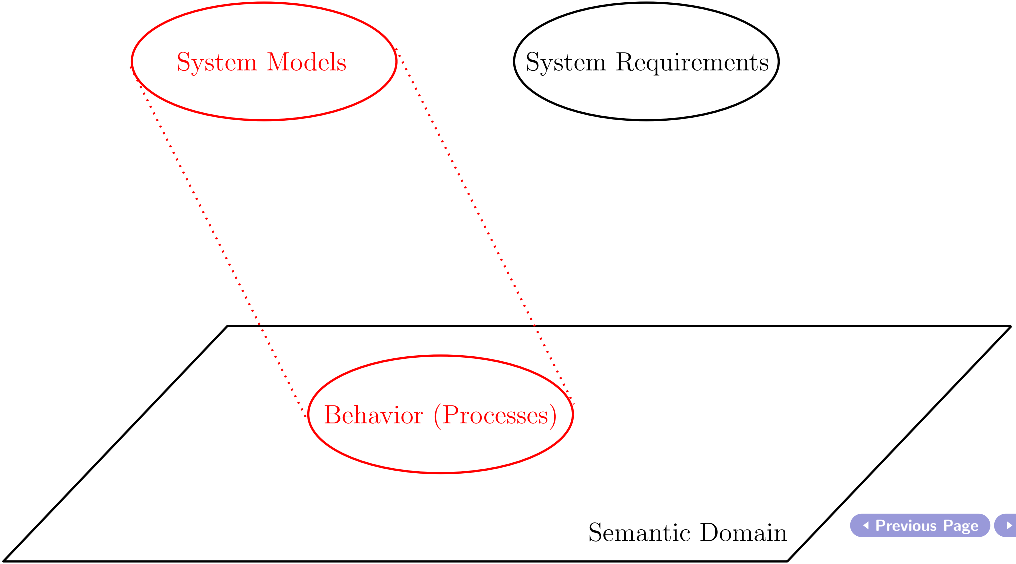
Constructed Types

Sets and Bags

- Syntax: sort S = Bag(St)
- Set enumeration: $\{a, b, \ldots\}$
- Bag enumeration: $\{a : 3, b : 2, \ldots\}$
- Several built-in constructs and maps

- Syntax: sort S $=$ Bag(St)
- Set enumeration: $\{a, b, \ldots\}$
- Bag enumeration: $\{a : 3, b : 2, \ldots\}$
- Several built-in constructs and maps
- Type casts: Set2Bag and Bag2Set

General Overview



Thank you very much.