System Validation

Mohammad Mousavi

3. Abstract Data Types

Abstract Data Types

Mohammad Mousavi

TU/Eindhoven

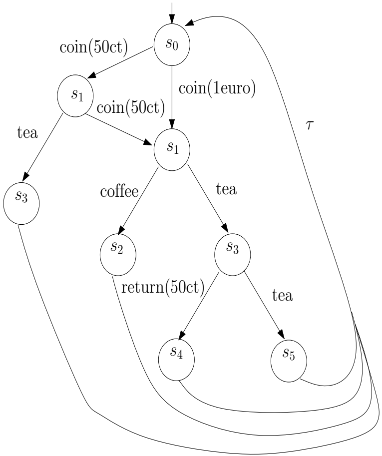
System Validation, 2012-2013
TU Delft

Overview

- Motivation
- Generic Concepts
- Other facilities
  1. built-in types,
  2. structured types,
  3. function types,
  4. constructed.

Motivating Example

Advanced coffee machine!



Generic Concepts

Data types

- Classes: sorts,

Generic Concepts

Data types

- Classes: sorts,
- Elements: constructors,

Generic Concepts

Data types

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

Euro Sort

```
sort Euro;
```

Euro Sort

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

Euro Sort

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

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

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)
       ...
```

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)
       ...
```

Theorem. zero $\neq$ one_euro.

Euro Sort

$$
\begin{array}{ll}
\texttt{sort} & \text{Euro;} \\
\texttt{cons} & \text{zero, fifty\_cents, one\_euro, more: Euro;} \\
 & \\
\text{map} & \text{eq: Euro} \times \text{Euro} \to \text{Bool;} \\
 & \text{plus: Euro} \times \text{Euro} \to \text{Euro;} \\
\text{var} & \text{e:Euro;} \\
\text{eqn} & \text{eq(e, e)= true;} \qquad (1) \\
 & \text{eq(zero, one\_euro)= false;} \qquad (2) \\
 & \text{eq(one\_euro, zero)= false;} \qquad (3) \\
 & \ldots
\end{array}
$$

Theorem. zero $\neq$ one_euro.

Proof by contradiction.

Euro Sort (Cont'd)

```
sort   Euro;
```

Euro Sort (Cont'd)

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

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;
```

iNatural

`sort` iNatural;

iNatural

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

iNatural

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

iNatural

`sort`	iNatural;	
`cons`	zero: iNatural;	
	succ: iNatural $\to$ iNatural;	
map	eq: iNatural $\times$ iNatural $\to$ Bool;	
var	i, j: iNatural;	
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)

iNatural

```
sort   iNatural;
cons   zero: iNatural;
       succ: iNatural → iNatural;
map    eq: iNatural × iNatural → Bool;
var    i, j: iNatural;
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)
```

Theorem. zero $\neq$ succ(i), for each iNatural i.

iNatural

$$
\begin{array}{ll}
\texttt{sort} & \text{iNatural;} \\
\texttt{cons} & \text{zero: iNatural;} \\
 & \text{succ: iNatural} \to \text{iNatural;} \\
\text{map} & \text{eq: iNatural} \times \text{iNatural} \to \text{Bool;} \\
\text{var} & \text{i, j: iNatural;} \\
\text{eqn} & \text{eq(i, i)= true;} \qquad (1) \\
 & \text{eq(zero, succ(i))= false;} \qquad (2) \\
 & \text{eq(succ(i), zero)= false;} \qquad (3) \\
 & \text{eq(succ(i), succ(j))= eq(i,j);} \qquad (4)
\end{array}
$$

Theorem. zero $\neq$ succ(i), for each iNatural i.

Proof by contradiction (see the next slide).

Proof of zero $\neq$ succ(i)

Assume towards contradiction that for some iNatural n, zero $=$ succ(n). Then, we have:

$$\begin{array}{lcl} \text{true} & = & (1) \\ \text{eq(zero, zero)} & = & \text{(assump.)} \\ \text{eq(zero, succ(n))} & = & (2) \\ \text{false} & & \end{array}$$

Induction

Proof Rule

Thesis: $P(s)$ for each s of a given sort S.

Rule:

- prove $P(c)$ for each constant c of sort S.
- assuming that $P(x_i)$ holds (induction hypothesis, for each $0 \leq i < n$), prove $P(f(x_0, \ldots, x_{n-1}))$ for each n-ary constructor of sort S.

Another Theorem

Theorem. eq(i, succ(i)) $=$ false, for each i.
Proof. By induction on i.

Another Theorem

Theorem. eq(i, succ(i)) $=$ false, for each i.
Proof. By induction on i.
Induction basis: $i =$ zero. It follows from axiom (1) that eq(zero, succ(zero)) $=$ false.
Induction hypothesis, i $=$ n: assume that n $\neq$ succ(n);
Induction step, i $=$ $succ(n)$: prove that succ(n) $\neq$ succ(succ(n)):

$$\begin{array}{lll} \text{eq(succ(n), succ(succ(n)))} & = & (4) \\ \text{eq(n, succ(n))} & = & \text{(ind. hyp.)} \\ \text{false} & & \end{array}$$

iNatural

`sort` iNatural

iNatural

$$\begin{array}{ll} \texttt{sort} & \text{iNatural} \\ \texttt{cons} & \text{zero : iNatural;} \\ & \text{succ: iNatural} \rightarrow \text{iNatural;} \end{array}$$

iNatural

`sort`	iNatural
`cons`	zero : iNatural;
	succ: iNatural $\to$ iNatural;
`map`	plus: iNatural $\times$ iNatural $\to$ iNatural;

iNatural

`sort`	iNatural	
`cons`	zero : iNatural;	
	succ: iNatural $\rightarrow$ iNatural;	
map	plus: iNatural $\times$ iNatural $\rightarrow$ iNatural;	
var	i,j :iNatural;	
eqn	plus(zero, i)= i;	(1)
	plus(i, zero)= i;	(2)
	plus(i, succ(j))= succ(plus(i, j));	(3)

iNatural

`sort`	iNatural
`cons`	zero : iNatural;
	succ: iNatural $\to$ iNatural;
map	plus: iNatural $\times$ iNatural $\to$ iNatural;
var	i,j :iNatural;
eqn	plus(zero, i)= i; (1)
	plus(i, zero)= i; (2)
	plus(i, succ(j))= succ(plus(i, j)); (3)

Theorem. plus(succ(i), j) $=$ succ(plus(i,j)), for each i.

iNatural

sort	iNatural	
cons	zero : iNatural;	
	succ: iNatural $\to$ iNatural;	
map	plus: iNatural $\times$ iNatural $\to$ iNatural;	
var	i,j :iNatural;	
eqn	plus(zero, i)= i;	(1)
	plus(i, zero)= i;	(2)
	plus(i, succ(j))= succ(plus(i, j));	(3)

Theorem. plus(succ(i), j) = succ(plus(i,j)), for each i.

Proof by induction on j (see the next slide).

plus(succ(i), j) = succ(plus(i,j))

Proof. By induction on j.

plus(succ(i), j) = succ(plus(i,j))

Proof. By induction on j.

Induction basis: j = zero:

$$
\begin{aligned}
\text{plus}(\text{succ}(i), \text{zero}) &= (2)\\
\text{succ(i)} &= (3, \text{ from right to left})\\
\text{succ}(\text{plus}(i, \text{zero}))
\end{aligned}
$$

Induction hypothesis, j = n: assume that plus(succ(i), n) = succ(plus(i,n);

Induction step, j = $succ(n)$: prove that plus(succ(i), succ(n)) = succ(plus(i, succ(n)):

$$
\begin{aligned}
\text{plus}(\text{succ}(i), \text{succ}(n)) &= (3)\\
\text{succ}(\text{plus}(\text{succ}(i), n)) &= (\text{ind. hyp.})\\
\text{succ}(\text{succ}(\text{plus}(i, n))) &= (3, \text{ from right to left})\\
\text{succ}(\text{plus}(i, \text{succ}(n)))
\end{aligned}
$$

ADT Facilities

Built-In Types

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

ADT Facilities

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.

ADT Facilities

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.

ADT Facilities

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

ADT Facilities

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.

ADT Facilities

Structured Types

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

ADT Facilities

Structured Types

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

ADT Facilities

Structured Types

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

```
sort  St
cons  elm_a, elm_b: St;
      f: St → St;
map   is_a, is_b, is_f: St → Bool;
```

ADT Facilities

Structured Types

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

```
sort   St
cons   elm_a, elm_b: St;
       f: St → St;
map    is_a, is_b, is_f: St → Bool;
var    s :St;
eqn    is_a(elm_a)= true;
       is_a(elm_b)= false;
       is_a(f(s))= false;
       ...
```

ADT Facilities

Structured Types

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

ADT Facilities

Constructed Types: Lists

- Syntax: sort lst = List(St);

ADT Facilities

Constructed Types: Lists

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

ADT Facilities

Constructed Types: Lists

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

ADT Facilities

Constructed Types: Lists

- 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.

ADT Facilities

Constructed Types: Sets and Bags

- Syntax: sort lst = Set(St);

ADT Facilities

Constructed Types: Sets and Bags

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

ADT Facilities

Constructed Types: Sets and Bags

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

ADT Facilities

Constructed Types: Sets and Bags

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

ADT Facilities

Constructed Types: Sets and Bags

- Syntax: sort lst = 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

Announcements

- The reader is available and can be ordered.
- Next lecture will be given by Jan Friso Groote.