Foundations of System Development

Martin Wirsing

in cooperation with
Axel Rauschmayer

WS 05/06

Goals

- Refinement of data structures:
  change of data structures

Change of Data Structures

"Simulation" of a Σ-structure through a Σ_1-structure.

- A (S, F)-structure A is simulated by a (S_1, F_1)-structure B if every carrier set A_s of A is represented by a subset
$$Rep_s \subseteq B_{s'}$$
of a carrier set $B_{s'}$ of B and if every function symbol $f \in F$ is represented by a function symbol $f_1 \in F_1$.
- Plenty elements of Rep_s can represent the same element of A_s. This induces a congruence relation $\sim_s$.
- Rep must be closed under the operations of F.
- $\sim_s$ must be compatible with the operations of A_s.

Simulation

Definition(Simulation):

1. Let $\Sigma \subseteq \Sigma_1$.
   A Σ_1-structure B simulates identically a Σ-structure A w.r.t. $Rep^B, \sim^B$, if
   (a) $Rep^B_s \subseteq B_s$ for all $s \in S$,
   (b) $\sim^B_s$ is a Σ-congruence on Rep^B_s for all $s \in S$, and
   (c) A is isomorphic to $Rep^B/\sim^B$ whereby
   $Rep^B/\sim^B =_{def} ((Rep^B_s)/\sim^B_s)_{s\in S}$.
2. A Σ_1-structure B simulates a Σ-structure A w.r.t. renaming $\rho : \Sigma \to \Sigma_1$, Rep^B and $\sim^B$ if
   (a) $Rep^B_s \subseteq B_{\rho(s)}$ for all $s \in S$,
   (b) $\sim^B_s$ is a $\rho(\Sigma)$-congruence on Rep^B_s for all $s \in S$, and
   (c) $A \cong Rep^B/\sim^B$.
   Therefore every identic simulation is a simulation. W.r.t. the inclusion $in : \Sigma \to \Sigma_1$.

Example: Sets by lists

- Consider the following signature of sets over natural numbers:

```
Sig(SETNAT) =
     including Sig(BOOL) .      including Sig(NATURAL) .
     sort Set .
     op empty : Set .
     op {_} : Nat -> Set .
     op add : Nat Set -> Set .
     op _∪_ : Set Set -> Set .
     op _in_ : Nat Set -> Bool .
```

- Let $P_{fin}(N)$ be the standard Sig(SETNAT) algebra of finite sets of natural numbers.
- Let Sig(SET-by-LIST) be a signature of lists over natural numbers which includes the operations of Sig(SETNAT); i.e. Sig(SET-by-LIST) has the form

```
     sorts Natural, List, … op empty: -> List . …
```

- Let ρ: Sig(SETNAT) -> Sig(SET-by-LIST) defined by

```
     sort Set to List .
```

  i.e. ρ(`Set`) = `List` and ρ(x) = x otherwise .

The structure N* of finite lists simulates the structure of finite sets $P_{fin}(N)$ on natural numbers w.r.t. the renaming ρ in the following different ways:

- Let U be the structure (N, N*, ε …) of (unordered) lists over natural numbers.
- Let G be the structure

  $(N, \{<x_1, \ldots, x_n> \mid x_1 < \ldots < x_n, n >= 0\}, \varepsilon \ldots)$

  of ordered lists.
- Let SG be the structure

  $(N, \{<x_1, \ldots, x_n> \mid x_1 <= \ldots <= x_n, n >= 0\}, \varepsilon \ldots)$

  of weakly ordered lists.

Simulation of Sets by Ordered Lists

$$
\begin{array}{rcl}
Rep^G_{\mathsf{Set}} & = & \{\langle x_1,\ldots,x_n\rangle | x_1 < \ldots < x_n, n \geq 0\} \\
\mathsf{empty}^G & = & \epsilon \\
x \ \mathsf{in}^G \ \langle x_1,\ldots,x_n\rangle & \Leftrightarrow & x = x_i \text{ for some } i \in \{1,\ldots,n\} \\
\{x\}^G & = & \langle x\rangle \\
\mathsf{add}^G(x,\langle x_1,\ldots,x_n\rangle) & = & \begin{cases} \langle x_1,\ldots,x_i,x,x_{i+1},\ldots,x_n\rangle \\ \quad \text{if } x_i < x < x_{i+1} \\ \langle x_1,\ldots,x_n\rangle \\ \quad \text{if } x = x_i \text{ for some } i \end{cases} \\
\langle x_1,\ldots,x_n\rangle \cup^G \langle y_1,\ldots,y_m\rangle & = & \langle z_1,\ldots,z_k\rangle \in Rep^G_{\mathsf{Set}}, \\
\text{whereby } \{x_1,\ldots,x_n,y_1,\ldots,y_m\} & = & \{z_1,\ldots,z_k\}
\end{array}
$$

$$
s_1 \sim^G s_2 \quad \Leftrightarrow \quad s_1 = s_2
$$

Simulation of Sets by Weakly Ordered Lists

$$
\begin{array}{rcl}
Rep^{SG}_{\mathsf{Set}} & = & \{\langle x_1,\ldots,x_n\rangle | x_1 \leq \ldots \leq x_n, n \geq 0\} \\
\mathsf{empty}^{SG} & = & \epsilon \\
\{x\}^{SG} & = & \langle\langle x\rangle \\
x \ \mathsf{in}^{SG} \ \langle x_1,\ldots,x_n\rangle & \Leftrightarrow & x = x_i \text{ for some } i \in \{1,\ldots,n\} \\
\mathsf{add}^{SG}(x,\langle x_1,\ldots,x_n\rangle) & = & \langle x_1,\ldots,x_i,x,x_i+1,\ldots,x_n\rangle \text{if} \\
 & & x_i \leq x \leq x_{i+1} \\
\langle x_1,\ldots,x_n\rangle \cup^{SG} \langle y_1,\ldots,y_m\rangle & = & \langle z_1,\ldots,z_k\rangle \in Rep^{SG}_{\mathsf{Set}} \\
 & & \text{whereby } \langle z_1,\ldots,z_k\rangle \text{ is} \\
 & & \text{a weakly ordered permutation of} \\
 & & \langle x_1,\ldots,x_n\rangle \mathbin{++} \langle y_1,\ldots,y_m\rangle
\end{array}
$$

$$
\langle x_1,\ldots,x_n\rangle \sim^{SG} \langle y_1,\ldots,y_m\rangle \quad \Leftrightarrow \quad \{x_1,\ldots,x_n\} = \{y_1,\ldots,y_m\},
$$

both sequences have the same elements

Constructing Simulations

The Forget-Restrict-Identify method for constructing simulations

1. Forget:

   Forget all symbols, that do not stem from $\rho(\Sigma)$.

2. Restrict:

   Restrict the carrier sets to the representing sets Rep_s.

3. Identify: Build the quotient w.r.t. $\sim_s$.

FRI-Implementation

Definition:

A specification SP_1 FRI-implements a specification SP w.r.t. a signature morphism $\sigma : \text{Sig}(SP) \to \text{Sig}(SP_1)$ (write $SP_1 \rightsquigarrow_\sigma SP$), if every model B of SP_1 simulates a model of SP w.r.t. suitable Rep^B and $\sim^B$.

Theorem:

The implementation relationship $\rightsquigarrow_\sigma$ is transitive: if $SP_1 \rightsquigarrow_{\sigma_1} SP_2$ and $SP_2 \rightsquigarrow_{\sigma_2} SP_3$ implies $SP_1 \rightsquigarrow_{\sigma_1 \circ \sigma_2} SP_3$.

Example: Implementing Stacks by Finite Maps (Arrays)

- **Maude specification of parameterized stacks:**

```
fmod STACK{X :: TRIV} is
   sorts Stack{X} NeStack{X} .
   subsort NeStack{X} < Stack{X} .
   op empty : -> Stack{X} [ctor] .
   op push : X$Elt Stack{X} -> NeStack{X} [ctor] .
   op pop : NeStack{X} -> NeStack{X} .
   op top : NeStack{X} -> X$Elt  .

   var E : X$Elt . var S : Stack{X} .

   eq top(push(E, S)) = E .
   eq pop(push(E, S)) = S .
endfm
```


Example: Implementing Stacks by Finite Maps (Arrays)

- **Instantiation of stacks by character symbols and natural numbers:**

```
view Char from TRIV to STRING is
   sort Elt to Char .
endv

fmod CHAR-STACK is
   including STACK{Char} .
endfm

view Natural from TRIV to NATURAL is
   sort Elt to Natural .
endv

fmod NAT-STACK is
   including STACK{Natural} .
endfm
```

Example: Implementing Stacks by Finite Maps (Arrays)

- **Simulating stacks over finite maps indexed by nat. numbers :**

```
fmod STACK-BY-MAP{X :: TRIV} is
  protecting MAP{Natural, X} .
  sorts Stack{X} NeStack{X} .
  subsort NeStack{X} < Stack{X} .
  op pair : Map{Natural, X} Natural -> Stack{X} [ctor] .
  op emptyStack : -> Stack{X} .
  op push : X$Elt Stack{X} -> NeStack{X} .
  op pop : NeStack{X} -> NeStack{X} .
  op top : NeStack{X} -> X$Elt  .

  var E : X$Elt . var I : Natural .
  var M : Map{Natural, X} .
  eq emptyStack = pair(empty, 0) .
  eq push(E, pair(M, I)) = pair(insert(I, E, M), s I) .
  eq top(pair(M, s I)) = M[I] .
  eq pop(pair(M, s I)) = pair(M, I) .
endfm
```


Example: Implementing Stacks by Finite Maps (Arrays)

- Instantiating maps by character symbols:

```
fmod CHAR-STACK-BY-MAP is
  including STACK-BY-MAP{Char} .
endfm

red in CHAR-STACK-BY-MAP :
  top(push("a", push("b", emptyStack))) .
red in CHAR-STACK-BY-MAP :
  top(pop(push("a", push("b", emptyStack)))) .
```

Example: Implementing Stacks by Finite Maps (Arrays)

Theorem

`STACK-BY-MAP{X :: TRIV}` is an

FRI-implementation of `STACK{X :: TRIV}`.

Proof:

Let M0 be any model of `STACK-BY-MAP` and restrict it to to the signature of STACK:

$$M = M0|_{sig(STACK\{X :: TRIV\})}$$

Define the following representation set and congruence:

$$Rep^M_{Elt} = M_{Elt}$$

$$Rep^M_{Stack\{X\}} = \{pair^M(m, i) \mid m \in M_{Map};\ i <= |m|+1\ \}$$

$$pair^M(m_1, i) \sim_M pair^M(m_2, j) \text{ iff}$$

$$i = j \wedge \forall k : Natural: k < i \Rightarrow m_1[k]^M = m_2[k]^M$$

- Then the quotient M' = $Rep_M/\sim_{M)}$, is a well-defined sig(STACK{X :: TRIV} algebra which is generated by empty and push.
- The two STACK axioms hold in M' :

1. M' |= top(push(x, pair(m, i))) = x :

Let v be any valuation. Then

M',v |= `top(push(x, pair(m, i)))` = [Def. of push]
`top(pair(insert(i, x, m), s i))` = [Def. of top]
`m[i]` = [Def. of _[_] in MAP]
x

2. M' |= pop(push(x, pair(m, i))) = pair(m, i) :

Let v be any valuation. Then

M',v |= `pop(push(x, pair(m, i)))` = [Def. of push]
`pop(pair(insert(i, x, m), s i))` = [Def. of pop]
`pair(insert(i, x, m), i)` =
[for all `k < i : insert(i,x,m)[k] = m[k]`]
`~ pair(m, i)` `q.e.d`

Theorem:

Let SP = (Σ,E) be a functional specification,
SP' a specification with Σ ⊂ Sig(SP') and let
Ax(Rep,) be an axiomatisation of

- a characteristic predicate Rep and
- a Σ-congruence relation ~ over SP'.

Let

```
fmod SP'' is
      protecting SP' .
      Ax(Rep,~) .
endfm
```

Then SP' is a FRI-Implementation of SP, if

- Rep/~ is freely generated by the Σ-constructors of SP'
- SP'' fulfils the axioms E of SP on Rep/~ , i.e.
  SP'' |= G_{Rep}, for all G ∈ E

whereby $G_{Rep,\sim}$ is defined inductively by:

$$
\begin{aligned}
p(t_1,\ldots,t_n)_{Rep,\sim} &\equiv p(t_1,\ldots,t_n)\\
(u=v)_{Rep,\sim} &\equiv u \sim v\\
(G_1 \wedge G_2)_{Rep,\sim} &\equiv (G_1)_{Rep,\sim} \wedge (G_2)_{Rep,\sim}\\
(\neg G)_{Rep,\sim} &\equiv \neg(G_{Rep,\sim})\\
(\forall x:s.\ G)_{Rep,\sim} &\equiv \forall x:s.\ Rep_s(x) \implies G_{Rep,\sim}\\
(\exists x:s.\ G)_{Rep,\sim} &\equiv \exists x:s.\ Rep_s(x) \wedge G_{Rep,\sim}
\end{aligned}
$$

Example: Implementing Stacks by Finite Maps (Arrays)

We define axiomatically the characteristic predicate Rep of the representation set and the congruence ~ over STACK-by-MAP:

```
Rep_Elt : X$Elt -> Bool .
Rep_Stack{X} : Stack{X} -> Bool .
_~_Elt_: X$Elt X$Elt -> Bool .
_~_Stack{X}_: Stack{X} Stack{X} -> Bool .
vars E, E' : X$Elt . var St : Stack{X} .
vars M, M' : Map{Nat, X} . vars  I, J : Natural .
eq Rep_Elt(E) = true .  ***Rep_Elt holds for all E ∈ X$Elt
eq Rep_Stack{X}(pair(M, I)) = true if I <= |M| + 1 .
eq  E ~_Elt E' = (E == E').
eq  pair(M, I) ~_Stack{X} pair(M', J)  =
     I == J and
     forall(k : Nat . k < I => M[k] == M'[k]) .
```


Then we can prove the STACK axioms as follows:

1. ∀ E: X$Elt . ∀ St : Stack{X} . top(push(E, St)) = E :

Relativization w.r.t. Rep and ~ yields

```
∀ E: X$Elt . ∀ St : Stack{X} .
   Rep_Elt(E) and Rep_Stack{X}(St) => top(push(E, St))) ~_Elt E .
```

By the definitions of Rep and ~ we get:

```
∀ E: X$Elt . ∀ M : Map{Nat, X} . ∀ I : Natural .
   (I <= |M|+1) => top(push(E, pair(M, I))) = E .
```

We prove this by the axioms of STACK-by-MAP:

```
top(push(E, pair(M, I))) =                [Def. of push]
top(pair(insert(I, E, M), s I)) =         [Def. of top]
M[I] =                                    [Def. of _[_] in MAP]
E
```

2. ∀ E: X$Elt . ∀ St : Stack{X} . pop(push(E, St))) = St:

Relativization w.r.t. Rep and ~ yields

∀ E: X$Elt . ∀ St : Stack{X} .

Rep_{Elt}(E) and $Rep_{Stack\{X\}}$(St) => pop(push(E, St))) $\sim_{Stack\{X\}}$ St By the definition of Rep we get:

∀ E: X$Elt . ∀ M : Map{Nat, X} . ∀ I : Natural .

(I <= |M|+1) =>

top(push(E, pair(M, I))) $\sim_{Stack\{X\}}$ pair(M, I))) .

We prove this by the axioms of STACK-by-MAP:

pop(push(E, pair(M, I))) = [Def. of push]

pop(pair(insert(I, E, M), s I)) = [Def. of pop]

pair(insert(I, E, M), I) $\sim_{Stack\{X\}}$

[for all k < I : insert(I,E,M)[k] = M[k]]

pair(M, I) q.e.d.

Summary

- If a Σ_1-algebra B simulates a Σ-algebra A as follows (called change of data structure):
  Every carrier set of A is represented by a subset Rep of a carrier set of B, and every function symbol of Σ is represented by a function symbol of Σ_1. Several elements of Rep can represent the same element of A, thus inducing an equivalence relation $\sim$
- A specification SP_1 FRI-implements a specification SP w.r.t. a signature morphism ρ , if every model of SP_1 simulates a model of SP w.r.t. suitable Rep and $\sim$.
- Implementation relationships are proved on the level of specifications. The characteristic predicate of Rep is used for this purpose. A specification SP' FRI-implements a specification SP, if Rep and $\sim$ can be defined over SP' in such a way that $E_{Rep,\sim}$ holds in SP' for any axiom E of SP.