

Computer Supported Modeling and Reasoning

David Basin, Achim D. Brucker, Jan-Georg Smaus, and
Burkhardt Wolff

April 2005

<http://www.infsec.ethz.ch/education/permanent/csmr/>

Higher-Order Logic Applications: Refinements

Burkhart Wolff

Overview

In previous weeks, we saw various **embeddings** in HOL:

Overview

In previous weeks, we saw various **embeddings** in HOL:

- Imperative languages

Overview

In previous weeks, we saw various **embeddings** in HOL:

- Imperative languages
- Functional languages

Overview

In previous weeks, we saw various **embeddings** in HOL:

- Imperative languages
- Functional languages
- Fragments of Specification Languages (HOL, Z)

Overview

In previous weeks, we saw various **embeddings** in HOL:

- Imperative languages
- Functional languages
- Fragments of Specification Languages (HOL, Z)

Can we apply these theories to **development methods** such as

Refinement ?

Overview

In previous weeks, we saw various **embeddings** in HOL:

- Imperative languages
- Functional languages
- Fragments of Specification Languages (HOL, Z)

Can we apply these theories to **development methods** such as

Refinement ?

Can we apply HOL to prove the relations **between** functions, operations, processes, architectures?

Rough Overview

Various Refinement Methods are described in the literature:

- Observational/Behavioural Equivalence

Rough Overview

Various Refinement Methods are described in the literature:

- Observational/Behavioural Equivalence
- Forget/Restrict/Identify-Method

Rough Overview

Various Refinement Methods are described in the literature:

- Observational/Behavioural Equivalence
- Forget/Restrict/Identify-Method
- Operation Refinement, Data Refinement [Spi92]

Rough Overview

Various Refinement Methods are described in the literature:

- Observational/Behavioural Equivalence
- Forget/Restrict/Identify-Method
- Operation Refinement, Data Refinement [Spi92]
- Refinement Calculus

Rough Overview

Various Refinement Methods are described in the literature:

- Observational/Behavioural Equivalence
- Forget/Restrict/Identify-Method
- Operation Refinement, Data Refinement [[Spi92](#)]
- Refinement Calculus
- Process Refinement (CSP [[A.W97](#)])

Rough Overview

Various Refinement Methods are described in the literature:

- Observational/Behavioural Equivalence
- Forget/Restrict/Identify-Method
- Operation Refinement, Data Refinement [Spi92]
- Refinement Calculus
- Process Refinement (CSP [A.W97])
- Machine Refinement (B-Method [Abr96])
-

(thousands of articles and many books on the subject.
Arbitrary selection by the author).

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

one system step involving
complex transformation of
input, output and state data,

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

one system step involving complex transformation of input, output and state data,

- Behavioral Modelling:

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

one system step involving complex transformation of input, output and state data,

- Behavioral Modelling:

sequences of system steps considering the evolution of input, output and states.

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

data refinement (Z, KIV, B),

algebraic specification techniques

(Behavioural Equivalence),

Hoare-like calculi (Morgan, Back/Wright)

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

data refinement(Z, KIV, B),

algebraic specification techniques

(Behavioural Equivalence),

Hoare-like calculi (Morgan, Back/Wright)

- Behavioral Modelling:

Common Formal Method Classification

One distinguishes:

- Data-Oriented Modelling Techniques:

data refinement (Z, KIV, B),

algebraic specification techniques

(Behavioural Equivalence),

Hoare-like calculi (Morgan, Back/Wright)

- Behavioral Modelling:

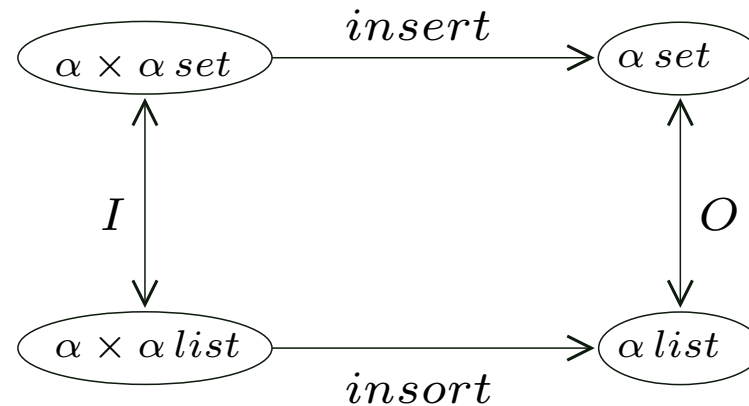
process algebras (CSP, CCS, . . .),

temporal logics

Data Refinement for a Function

A simple example for refining a function:

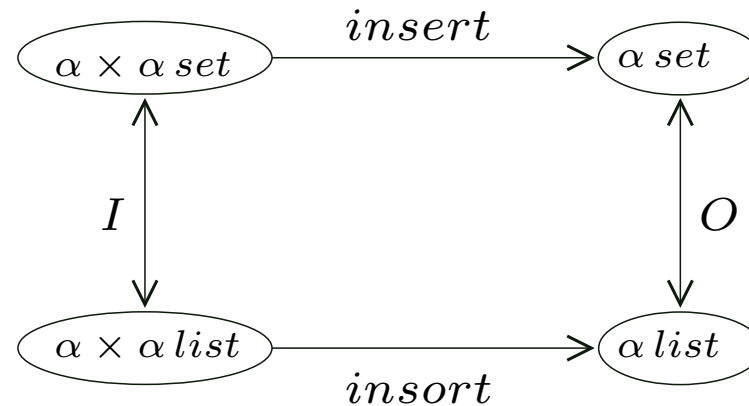
Representing Sets by Lists



Data Refinement for a Function

A simple example for refining a function:

Representing Sets by Lists



Can this be generalized to **operations** (i.e. “procedures” with input, output, and an implicit state transition) ?

Data Refinement

Principles of Data-Refinement

- Forward Simulation

Data Refinement

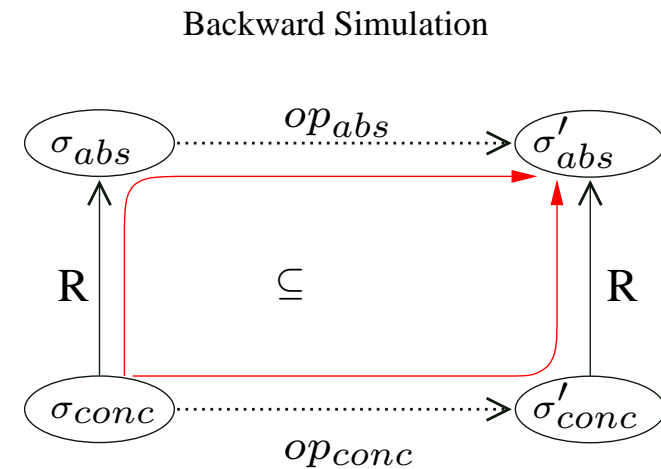
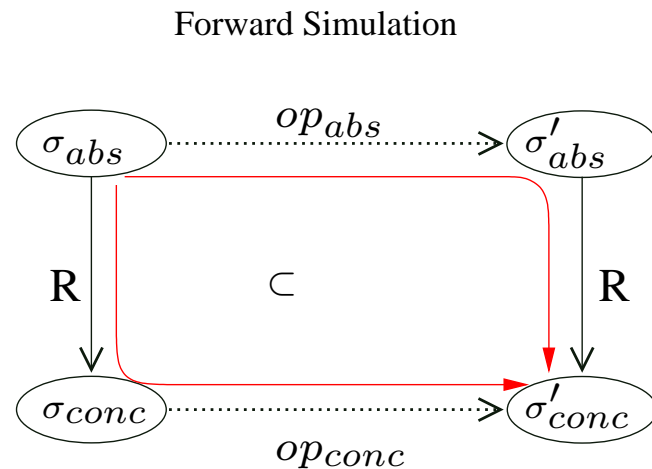
Principles of Data-Refinement

- Forward Simulation
- Backward Simulation

Data Refinement

Principles of Data-Refinement

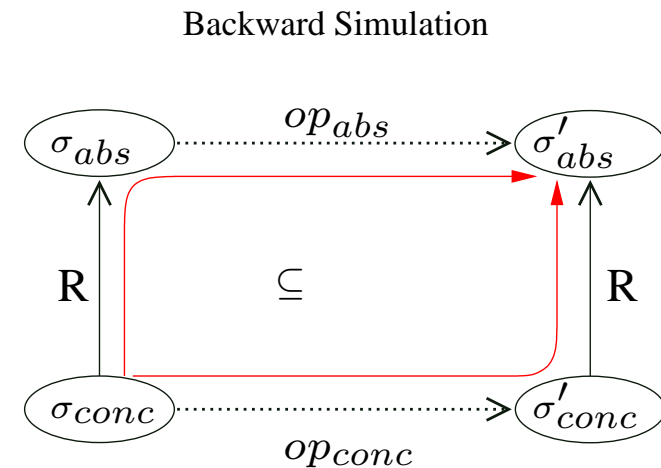
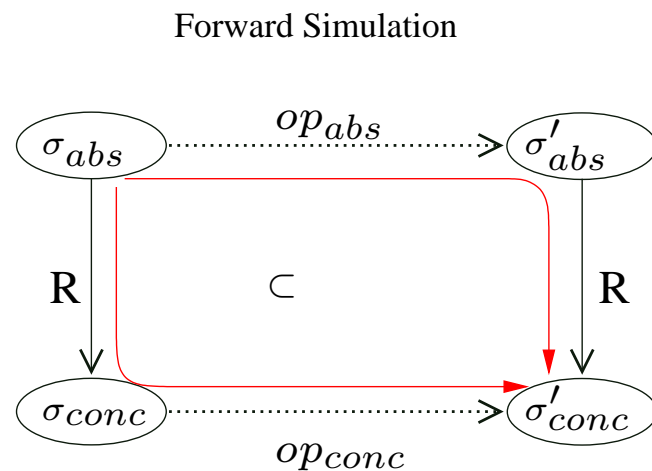
- Forward Simulation
- Backward Simulation



Data Refinement

Principles of Data-Refinement

- Forward Simulation
- Backward Simulation



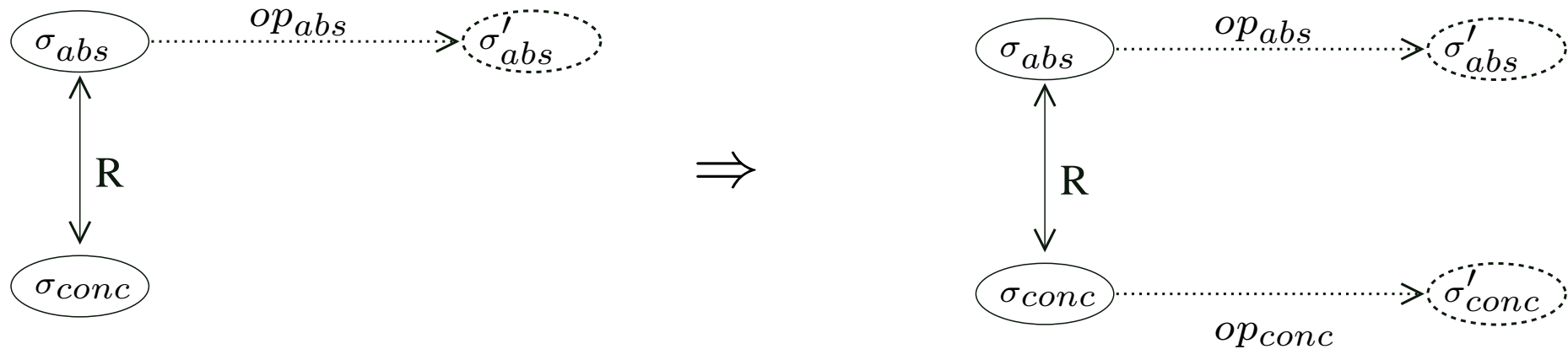
See also [Spi92] and [WD96]!

Data Refinement

Forward Simulation

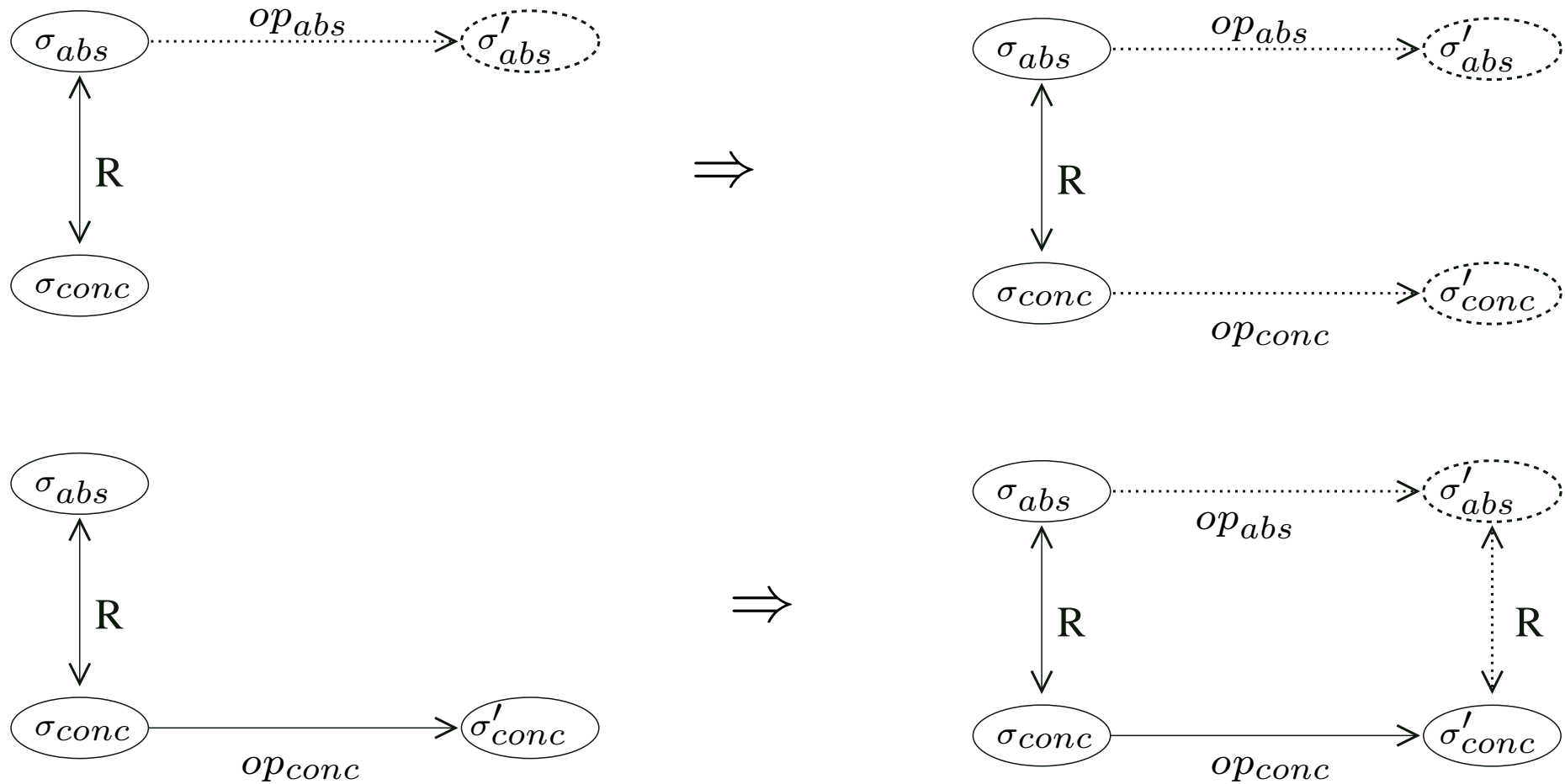
Data Refinement

Forward Simulation



Data Refinement

Forward Simulation



Data Refinement

Can we

Data Refinement

Can we

- represent refinement in Isabelle ?

Data Refinement

Can we

- represent refinement in Isabelle ?
- verify and compare refinement notions ?

Data Refinement

Can we

- represent refinement in Isabelle ?
- verify and compare refinement notions ?
- integrate refinement for functions and operations?

Data Refinement

Can we

- represent refinement in Isabelle ?
- verify and compare refinement notions ?
- integrate refinement for functions and operations?

YES! In the following, we present a theory of **Abstract IOS Specifications** and a forward simulation refinement on it.
(backward refinement is analogously)

IOS-Forward Simulation

An abstract system IOS-step has the type:

IOS-Forward Simulation

An abstract system IOS-step has the type:

types ('i, 'o, 's) ios_rel = " (('i × 's) × ('o × 's)) set "

An Abstract IOS Specification is:

(closely related to a Z operation schema):

```
record ('i, 'o, 's) spec =  
  init  :: "'s set"  
  inv   :: "'s set"  
  opn   :: "' ('i, 'o, 's) ios_rel "
```

IOS-Forward Simulation

The **generalized abstraction relation** on abstract IOS specifications looks as follows:

```
record ('i','i ','o','o ','s','s') abs_rel =  
  i      :: "'i × 'i' set"  
  o      :: "'o × 'o' set"  
  abs    :: "'s × 's' set"
```

The relation is just a triple of relations on input data, output data and states.

IOS-Forward Simulation

We define a FS-refinement on IOS specifications by its three “proof obligations” :

IOS-Forward Simulation

We define a FS-refinement on IOS specifications by its three “proof obligations”:

constdefs

$$\text{FS_refine} \quad :: \quad " [('i, 'o, 's) \text{ spec},$$

$$\quad \quad \quad ('i, 'i', 'o, 'o', 's, 's') \text{ abs_rel},$$

$$\quad \quad \quad ('i', 'o', 's') \text{ spec}] \Rightarrow \text{bool}"$$

$$A \sqsubseteq_{\text{FS}} R C \equiv$$

$$\text{FS_init } A R C \wedge \text{FS_corr1 } A R C \wedge \text{FS_corr2 } A R C$$

IOS-Forward Simulation

We define a FS-refinement on IOS specifications by its three “proof obligations”:

constdefs

$$\text{FS_refine} ::= \text{'' } [(\text{' i , ' o , ' s }) \text{ spec,} \\ \text{(\text{' i , ' i ' , ' o , ' o ' , ' s , ' s ' }) \text{ abs_rel ,} \\ \text{(\text{' i ' , ' o ' , ' s ' }) \text{ spec}] \Rightarrow \text{bool''}$$

$$A \sqsubseteq_{R} C \equiv$$

$$\text{FS_init } A \text{ R } C \wedge \text{FS_corr1 } A \text{ R } C \wedge \text{FS_corr2 } A \text{ R } C$$

In conceptual notation, we will also write $A \sqsubseteq_{R}^{fs} B$ for forward simulation (resp. $A \sqsubseteq_{R}^{bs} B$ for backward simulation).

IOS-Forward Simulation

The three conditions are:

IOS-Forward Simulation

The three conditions are:

- **FS_init**: The set of initial states must be compatible,
- **FS_corr2**: When an abstract state transition is possible, then a corresponding concrete state transition must be possible,
- **FS_corr1**: When a concrete operation reaches a target state, then the corresponding abstract must exist.

IOS-Forward Simulation

The three conditions are:

- **FS_init**: The set of initial states must be compatible,
- **FS_corr2**: When an abstract state transition is possible, then a corresponding concrete state transition must be possible,
- **FS_corr1**: When a concrete operation reaches a target state, then the corresponding abstract must exist.

(Terminology follows [WD96]).

IOS-Forward Simulation

The proof-obligation `FS_init`

IOS-Forward Simulation

The proof-obligation FS_init

$FS_init\ A\ R\ C \equiv$

$\forall cs \in (inv\ C). cs \in (init\ C) \longrightarrow$

$\exists as \in (inv\ A). as \in (init\ A) \wedge (as, cs) \in abs\ R$

IOS-Forward Simulation

The proof-obligation FS_init

$FS_init\ A\ R\ C \equiv$

$\forall cs \in (inv\ C). cs \in (init\ C) \longrightarrow$

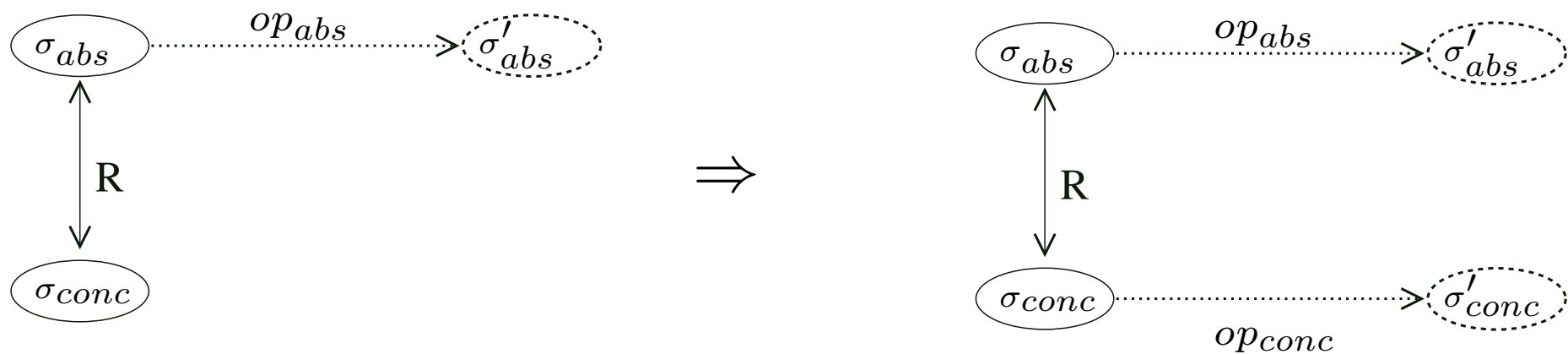
$\exists as \in (inv\ A). as \in (init\ A) \wedge (as, cs) \in abs\ R$

IOS-Forward Simulation

Recall the diagrams for FS_corr2

IOS-Forward Simulation

Recall the diagrams for FS_corr2



IOS-Forward Simulation

The formalization for FS_corr2

IOS-Forward Simulation

The formalization for FS_corr2

FS_corr2 A R C \equiv

$\forall as \in (\text{inv } A). \forall cs \in (\text{inv } C).$

$\forall inp \in (\text{Domain}(i \ R)). \forall inp' \in (\text{Range}(i \ R)).$

$((inp, as) \in \text{Domain}(\text{opn } A) \wedge$

$(as, cs) \in \text{abs } R \wedge (inp, inp') \in i \ R)$

\longrightarrow

$(inp', cs) \in \text{Domain}(\text{opn } C)$

IOS-Forward Simulation

The formalization for FS_corr2

FS_corr2 A R C \equiv

$\forall as \in (\text{inv } A). \forall cs \in (\text{inv } C).$

$\forall inp \in (\text{Domain}(i \ R)). \forall inp' \in (\text{Range}(i \ R)).$

$((inp, as) \in \text{Domain}(\text{opn } A) \wedge$

$(as, cs) \in \text{abs } R \wedge (inp, inp') \in i \ R)$

\longrightarrow

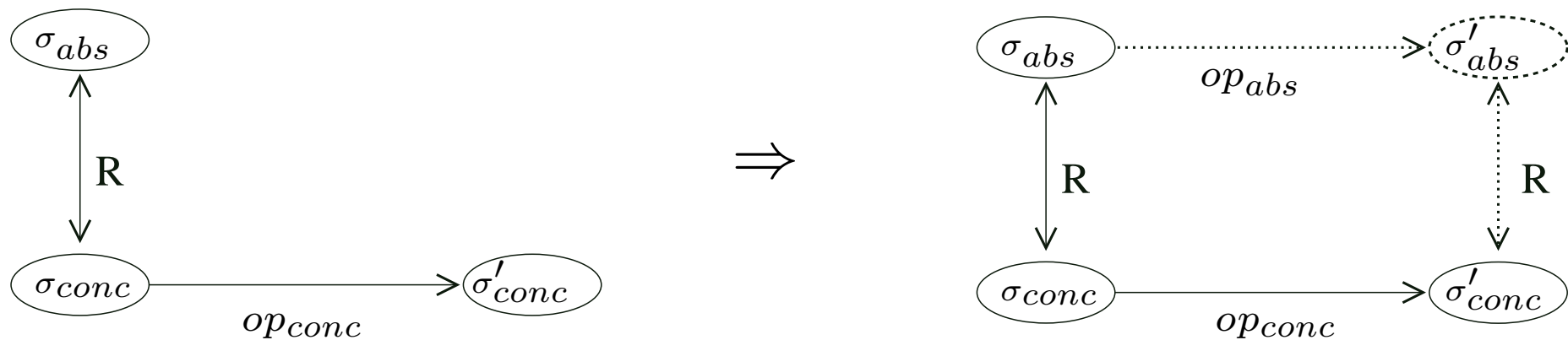
$(inp', cs) \in \text{Domain}(\text{opn } C)$

IOS-Forward Simulation

Recall the diagrams for FS_corr1

IOS-Forward Simulation

Recall the diagrams for FS_corr1



IOS-Forward Simulation

Recall the diagrams for FS_corr1

IOS-Forward Simulation

Recall the diagrams for FS_corr1

FS_corr1 A R C \equiv

$$\forall as \in (\text{inv } A). \forall cs \in (\text{inv } C). \forall cs' \in (\text{inv } C).$$

$$\forall inp \in (\text{Domain}(i \ R)). \forall inp' \in (\text{Range}(i \ R)). \forall out' \in (\text{Range}(o \ R)).$$

$$((inp, as) \in \text{Domain}(\text{opn } A) \wedge$$

$$(as, cs) \in \text{abs } R \wedge (inp, inp') \in i \ R \wedge$$

$$((inp', cs), (out', cs')) \in \text{opn } C)$$

\longrightarrow

$$(\exists as' \in (\text{inv } A). \exists out \in (\text{Domain}(o \ R)).$$

$$(as', cs') \in \text{abs } R \wedge (out, out') \in o \ R \wedge$$

$$((inp, as), (out, as')) \in \text{opn } A)$$

Tayloring IOS-Forward Simulation (1)

Tayloring forward simulations **for functions** :

Tayloring IOS-Forward Simulation (1)

Tayloring forward simulations **for functions** : Prerequisite:

We embed functions as abstract specifications:

Tayloring IOS-Forward Simulation (1)

Tayloring forward simulations **for functions** : Prerequisite:
We embed functions as abstract specifications:

constdefs

```
fun2op :: "'i set, 'i ⇒ 'o] ⇒ ('i, 'o, unit) spec"
"fun2op precondition F ≡ (| init = {()}, inv = {()},
                           opn = {(a,b). ∃ x∈precond. a=(x,()) ∧
                                   b=(F x,())})|)"
```

Tayloring IOS-Forward Simulation (1)

Tayloring forward simulations **for functions** : Prerequisite:
We embed functions as abstract specifications:

constdefs

```
fun2op :: "'i set, 'i ⇒ 'o] ⇒ ('i, 'o, unit) spec"
"fun2op precondition F ≡ (| init = {()}, inv = {()},
                          opn = {(a,b). ∃ x∈precond. a=(x,()) ∧
                                   b=(F x,())})|)"
```

precond serves as an additional means to formalize preconditions, under which the refinement is supposed to hold.

Tayloring IOS-Forward Simulation (1)

. . . derive the specialized version `FS_refine_fun` :

Tayloring IOS-Forward Simulation (1)

. . . derive the specialized version FS_refine_fun :

$\llbracket R = (\lvert i = RI, o = RO, abs = Id \rvert);$

$\forall inp \in pa. A\ inp \in \text{Domain } RO;$

$\forall inp \in pa. \forall inp'. (inp, inp') \in RI \longrightarrow inp' \in pc;$

$\forall inp \in pa. \forall inp' \in pc. (A\ inp, C\ inp') \in RO \rrbracket$

$\implies (\text{fun2op } pa\ A) \ \langle\text{sqsubseteq}\rangle\ R\ (\text{fun2op } pc\ C)''$

Tayloring IOS-Forward Simulation (1)

. . . derive the specialized version `FS_refine_fun`:

$\llbracket R = (\text{i} = \text{RI}, \text{o} = \text{RO}, \text{abs} = \text{Id}) \rrbracket$;

$\forall \text{inp} \in \text{pa}. A \text{ inp} \in \text{Domain RO};$

$\forall \text{inp} \in \text{pa}. \forall \text{inp}' . (\text{inp}, \text{inp}') \in \text{RI} \longrightarrow \text{inp}' \in \text{pc};$

$\forall \text{inp} \in \text{pa}. \forall \text{inp}' \in \text{pc}. (A \text{ inp}, C \text{ inp}') \in \text{RO} \rrbracket$

$\implies (\text{fun2op pa } A) \sqsubseteq \text{R} (\text{fun2op pc } C)$ "

Note that the first assumption constrains the structure of the generalized abstraction to default values on dummy states . . .

Tayloring IOS-Forward Simulation (1)

A (standard) example. We assume the usual:

Tayloring IOS-Forward Simulation (1)

A (standard) example. We assume the usual:

consts

insert :: " 'a :: order, 'a list] \Rightarrow 'a list "

is_sorted :: " 'a list] \Rightarrow bool"

Tayloring IOS-Forward Simulation (1)

A (standard) example. We assume the usual:

consts

```
insert    :: " ['a :: order, 'a list ]  $\Rightarrow$  'a list "  
is_sorted :: " ['a list ]  $\Rightarrow$  bool"
```

... and set up the refinement relation as:

consts

```
data_R    :: " ('a :: order set  $\times$  'a list )set"  
set_list_R :: " ('a :: order  $\times$  'a set , 'a  $\times$  'a list ,  
                'a set , 'a list ,  
                unit , unit) abs_rel"
```

defs

```
data_R_def:  "data_R  $\equiv$  {(x,y). x=set y  $\wedge$  is_sorted y}"
set_list_R_def :
    " set_list_R  $\equiv$  ( $\mid$ i = {(x,y). fst x = fst y  $\wedge$ 
                                (snd x, snd y)  $\in$  data_R},
    o = data_R, abs = Id)"
```

defs

```
data_R_def:  "data_R  $\equiv$  {(x,y). x=set y  $\wedge$  is_sorted y}"
set_list_R_def :
    " set_list_R  $\equiv$  (|i = {(x,y). fst x = fst y  $\wedge$ 
                        (snd x, snd y)  $\in$  data_R},
                      o = data_R, abs = Id)"
```

Tayloring IOS-Forward Simulation (1)

A refinement proof is started:

Tayloring IOS-Forward Simulation (1)

A refinement proof is started:

lemma insert_insort_refine_FS :

$$\begin{aligned} & \text{"(fun2op } \{\lambda(x,S). \text{ finite } S\} \quad (\lambda(x,S). \text{ insert } x \ S)) \\ & \quad \backslash \langle \text{sqsubseq} \rangle \text{set_list_R} \\ & \quad \text{(fun2op } \{\lambda(x,S). \text{ is_sorted } S\} \ (\lambda(x,S). \text{ insort } x \ S))\text{"} \end{aligned}$$

Tayloring IOS-Forward Simulation (1)

A refinement proof is started:

lemma insert_insort_refine_FS :

$$\begin{aligned} & \text{"(fun2op } \{\lambda(x,S). \text{ finite } S\} \quad (\lambda(x,S). \text{ insert } x \ S)) \\ & \quad \backslash \langle \text{sqsubseq} \rangle \text{set_list_R} \\ & \quad \text{(fun2op } \{\lambda(x,S). \text{ is_sorted } S\} \ (\lambda(x,S). \text{ insort } x \ S))\text{"} \end{aligned}$$

. . . and, after applying `FS_refine_fun` as introduction rule, we derive the proof obligations:

Tayloring IOS-Forward Simulation (1)

A refinement proof is started:

lemma `insert_insort_refine_FS` :

$$\begin{aligned} & \text{"(fun2op } \{\lambda(x,S). \text{ finite } S\} \quad (\lambda(x,S). \text{ insert } x \ S)) \\ & \quad \backslash \langle \text{sqsubseq} \rangle \text{set_list_R} \\ & \quad \text{(fun2op } \{\lambda(x,S). \text{ is_sorted } S\} \ (\lambda(x,S). \text{ insort } x \ S))\text{"} \end{aligned}$$

. . . and, after applying `FS_refine_fun` as introduction rule, we derive the proof obligations:

1. $\forall a \ b. \text{ finite } b \longrightarrow (\exists y. \text{ insert } a \ b = \text{ set } y \wedge \text{ is_sorted } y)$
2. $\forall a \ b. \text{ finite } b \longrightarrow$
 $(\forall aa \ ba.$
 $\quad \text{is_sorted } ba \longrightarrow \text{ insert } a \ b = \text{ set } (\text{insort } aa \ ba) \wedge$
 $\quad \text{is_sorted } (\text{insort } aa \ ba))$

Tayloring IOS-Forward Simulation (2)

. . . derive $FS_refine_opn_Z$ for operations

Tayloring IOS-Forward Simulation (2)

. . . derive FS_refine_opn_Z for operations

$\llbracket R = (\text{i} = \text{Id}, \text{o} = \text{Id}, \text{abs} = \text{Abs});$

$\forall \text{cs} \in (\text{inv } C). \text{cs} \in (\text{init } C) \longrightarrow$

$\exists \text{as} \in (\text{inv } A). \text{as} \in (\text{init } A) \wedge (\text{as}, \text{cs}) \in \text{Abs};$

$\forall \text{as} \in (\text{inv } A). \forall \text{cs} \in (\text{inv } C). \forall \text{inp} \in (\text{Domain}(\text{i } R)).$

$(\text{pre}(\text{opn } A)(\text{inp}, \text{as}) \wedge (\text{as}, \text{cs}) \in (\text{abs } R)) \longrightarrow$

$\text{pre}(\text{opn } C)(\text{inp}, \text{cs});$

$\forall \text{as} \in (\text{inv } A). \forall \text{cs} \in (\text{inv } C). \forall \text{cs}' \in (\text{inv } C). \forall \text{inp}. \forall \text{out}.$

$(\text{pre}(\text{opn } A)(\text{inp}, \text{as}) \wedge$

$(\text{as}, \text{cs}) \in \text{Abs} \wedge ((\text{inp}, \text{cs}), (\text{out}, \text{cs}')) \in \text{opn } C) \longrightarrow$

$\exists \text{as}' \in (\text{inv } A). (\text{as}', \text{cs}') \in \text{Abs} \wedge$

$((\text{inp}, \text{as}), (\text{out}, \text{as}')) \in (\text{opn } A) \rrbracket$

$\implies A \sqsubseteq_{\text{FS}} R C$

Tayloring IOS-Forward Simulation (2)

Do you recognize the pattern? :

Tayloring IOS-Forward Simulation (2)

Do you recognize the pattern? :This represents forward simulation a la [Spi92] and [WD96]):

Tayloring IOS-Forward Simulation (2)

Do you recognize the pattern? : This represents forward simulation a la [Spi92] and [WD96]):

$$\forall Cstate \bullet Cinit \rightarrow (\exists Astate \bullet Abs \wedge Ainit)$$

$$\forall Astate Cstate Cstate' x? y! \bullet$$

$$pre Aop \wedge Abs \wedge Cop \rightarrow (\exists Astate' \bullet Abs' \wedge Aop)$$

$$\forall Astate Cstate x? \bullet pre Aop \wedge Abs \rightarrow pre Cop$$

Note that in this refinement notion, input $x?$ and output $y!$ are identical!

Example: BirthdayBook Refinement

A (standard) example: Spivey's Birthdaybook[Spi92]:

Example: BirthdayBook Refinement

A (standard) example: Spivey's Birthdaybook[Spi92]: The states of the two systems are:

Example: BirthdayBook Refinement

A (standard) example: Spivey's Birthdaybook[Spi92]: The states of the two systems are:

```
record BirthdayBook =  
  birthday  :: "Name  $\leadsto$  Date"  
  known    :: "Name set"
```

```
record BirthdayBook1 =  
  dates     :: "(nat  $\leadsto$  Date)"  
  hwm      :: nat  
  names     :: "nat  $\leadsto$  Name"
```

(The invariant states that known is equal to the domain of birthday).

Example: BirthdayBook Refinement

The two operation schemas are immediately represented as abstract IOS specifications:

consts

```
AddBirthday :: "((Name × Date), unit, BirthdayBook) spec"
```

```
AddBirthday1:: "((Name × Date), unit, BirthdayBook1) spec"
```

```
...
```

Example: BirthdayBook Refinement

The abstraction relation between the underlying states is:

Example: BirthdayBook Refinement

The abstraction relation between the underlying states is:

constdefs

Abs :: "(BirthdayBook × BirthdayBook1) set"

"Abs ≡ {(x,y).((known x) = {n. ∃ i∈{1..(hwm y)}.
n = the (names y i)}) ∧
(∀ i∈{1..(hwm y)}. birthday x (the(names y i))
= dates y (the(names y i))))}"

Example: BirthdayBook Refinement

The abstraction relation between the underlying states is:

constdefs

Abs :: "(BirthdayBook × BirthdayBook1) set"

"Abs ≡ {(x, y). ((known x) = {n. ∃ i ∈ {1..(hwm y)}.
n = the (names y i)}) ∧
(∀ i ∈ {1..(hwm y)}. birthday x (the (names y i))
= dates y (the (names y i)))}"

. . . which is generalized to:

Example: BirthdayBook Refinement

The abstraction relation between the underlying states is:

constdefs

Abs :: "(BirthdayBook × BirthdayBook1) set"

"Abs ≡ {(x, y). ((known x) = {n. ∃ i ∈ {1..(hwm y)}.
 n = the (names y i)}) ∧
 (∀ i ∈ {1..(hwm y)}. birthday x (the (names y i))
 = dates y (the (names y i)))})}"

. . . which is generalized to:

constdefs

gen_Abs :: "('a, 'a, 'b, 'b, BirthdayBook, BirthdayBook1) abs_rel"

"gen_Abs ≡ (| i = Id, o = Id, abs = Abs)"

Example: BirthdayBook Refinement

The question to be asked:

Example: BirthdayBook Refinement

The question to be asked:

lemma AddBrithday__FS_refine :

"AddBirthday \<sqsubseteq>gen_Abs AddBirthday1"

Example: BirthdayBook Refinement

Applying `FS_refine_opn_Z` yields:

Example: BirthdayBook Refinement

Applying FS_refine_opn_Z yields:

1. $\forall cs \in \text{spec.inv AddBirthday1}.$
 $cs \in \text{init AddBirthday1} \longrightarrow$
 $(\exists as \in \text{inv AddBirthday}. as \in \text{init AddBirthday} \wedge (as, cs) \in \text{Abs})$
2. $\forall as \in \text{inv AddBirthday}. \forall cs \in \text{inv AddBirthday1}. \forall \text{inp}.$
 $\text{pre}(\text{opn AddBirthday})(\text{inp}, as) \wedge (as, cs) \in \text{Abs} \longrightarrow$
 $\text{pre}(\text{opn AddBirthday1})(\text{inp}, cs)$
3. $\forall as \in \text{inv AddBirthday}. \forall cs \in \text{inv AddBirthday1}.$
 $\forall cs' \in \text{inv AddBirthday1}. \forall \text{inp out}.$
 $\text{pre}(\text{opn AddBirthday})(\text{inp}, as) \wedge$
 $(as, cs) \in \text{Abs} \wedge ((\text{inp}, cs), \text{out}, cs') \in \text{opn AddBirthday1} \longrightarrow$
 $\exists as' \in \text{inv AddBirthday}.$
 $(as', cs') \in \text{Abs} \wedge ((\text{inp}, as), \text{out}, as') \in \text{opn AddBirthday}$

(see [Spi92] and the HOL-Z-distribution [BRW03]!)

Connection to Behavioral Refinement(1)

- How do abstract IOS specifications relate to behavioral models?

Connection to Behavioral Refinement(1)

- How do abstract IOS specifications relate to behavioral models?
- Can we extend reasoning over refinements of individual system steps to sequences of steps ?

Connection to Behavioral Refinement(1)

- How do abstract IOS specifications relate to behavioral models?
- Can we extend reasoning over refinements of individual system steps to sequences of steps ?
- How do established notions of behavioral specification relate to forward/backward simulation ?

Connection to Behavioral Refinement(1)

- How do abstract IOS specifications relate to behavioral models?
- Can we extend reasoning over refinements of individual system steps to sequences of steps ?
- How do established notions of behavioral specification relate to forward/backward simulation ?

Partial Answer: abstract IOS specifications generate behavioral notions like **Kripke-Structures**, **(Event) Traces** and **(Event) Failures**. The former talks about states, the latter two over “observable input/output” (=Events)

Connection to Behavioral Refinement(1)

State Projection into Kripke Structures :

Connection to Behavioral Refinement(1)

State Projection into Kripke Structures :

types

's trace = "nat \Rightarrow 's"

record 's kripke =

init :: "'s set"

step :: "'s \times 's set"

constdefs

state_projection :: "('i, 'o, 's) spec \Rightarrow 's kripke"

" state_projection A \equiv

(| kripke . init = spec . init A,

kripke . step = {(s1,s2). \exists i' o'. ((i', s1), (o', s2)) \in spec.opn A} |)"

Connection to Behavioral Refinement(1)

constdefs

```
is_trace      :: "'s kripke, 's trace] => bool"
" is_trace K t ≡ t 0 ∈ kripke . init K ∧
    (∀i. (t i, t (Suc i)) ∈ kripke . step K)"
traces       :: "'s kripke => 's trace set"
" traces K   ≡ { t. is_trace K t }"
```

Connection to Behavioral Refinement(1)

constdefs

```

is_trace      :: "'s kripke, 's trace] => bool"
" is_trace K t ≡ t 0 ∈ kripke . init K ∧
                (∀i. (t i, t (Suc i)) ∈ kripke . step K)"
traces       :: "'s kripke => 's trace set"
" traces K    ≡ { t. is_trace K t }"

```

And now, a standard temporal logics $K \models \phi$ can be defined on top of the Kripke structure K . **Open problem:** Under which conditions can a forward refinement allow for system abstractions?

$$\llbracket A \sqsubseteq R C; \text{kripke_projection } A \models \phi \rrbracket \\ \implies \text{kripke_projection } C \models \phi$$

Paves the way for temporal abstractions and model-checking.

Connection to Behavioral Refinement(2)

Event Projection of ('i, 'o, 's)spec's to event traces:

Connection to Behavioral Refinement(2)

Event Projection of $(\text{'i}, \text{'o}, \text{'s})\text{spec}$'s to event traces:

constdefs

`is_gen_trace` $:: [(\text{'i}, \text{'o}, \text{'s})\text{spec}, ((\text{'i} \times \text{'s}) \times (\text{'o} \times \text{'s}))\text{trace}] \Rightarrow \text{bool}$

`" is_gen_trace A t \equiv (snd (fst (t 0)) \in spec. init A \wedge
($\forall i. t\ i \in$ spec.opn A) \wedge
($\forall i. \text{snd}(\text{snd}(t\ i)) = \text{snd}(\text{fst}(t\ (\text{Suc } i)))$)"`

`gen_traces` $:: (\text{'i}, \text{'o}, \text{'s})\text{spec} \Rightarrow ((\text{'i} \times \text{'s}) \times (\text{'o} \times \text{'s}))\text{trace set}$

`" gen_traces A \equiv { t. is_gen_trace A t }`"

`event_traces_projection` $:: (\text{'i}, \text{'o}, \text{'s})\text{spec} \Rightarrow (\text{'i} \times \text{'o})\text{trace set}$

`" event_traces_projection A \equiv ($\lambda f\ n. (\lambda(x,y). (\text{fst } x, \text{fst } y))(f\ n)$)
‘ (gen_traces A) ”`

In order to accommodate ('i', 'o', 's)spec for CSP-processes, for example, states 's must be instantiated with process terms, and opn by transition of operational CSP semantics [A.W97] ...

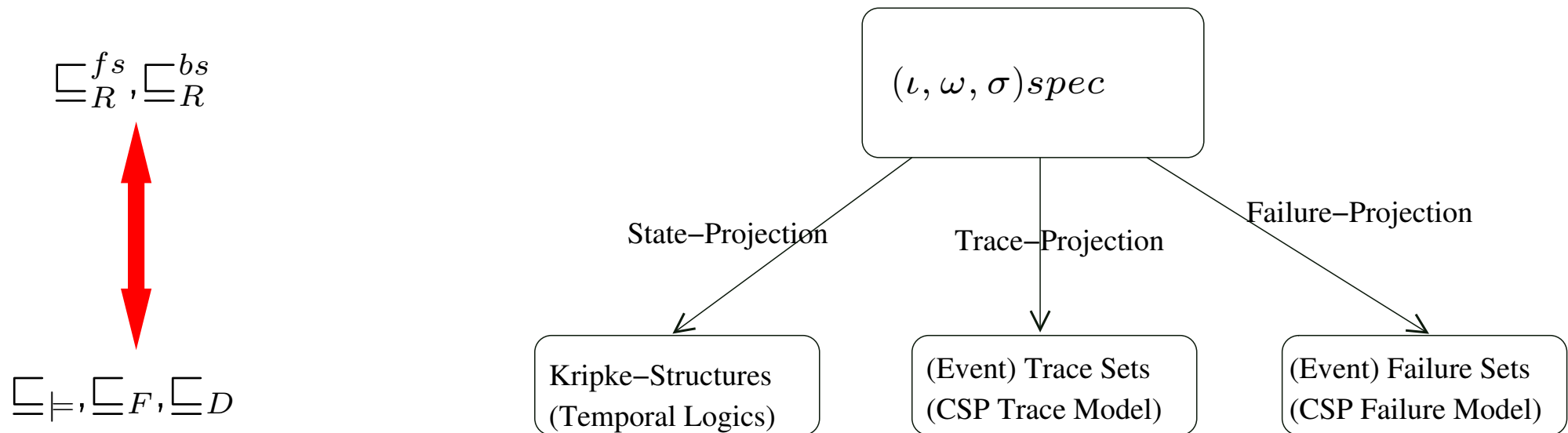
In order to accommodate ('i', 'o', 's)spec for CSP-processes, for example, states 's must be instantiated with process terms, and opn by transition of operational CSP semantics [A.W97] ...

Open problem: Under which conditions allows fs-refinement simplifying process-refinement, i.e. to inclusion of trace or failure sets?

Connection to Behavioral Refinement(3)

Overview:

Abstract Specifications and their Behavioral Models



Summary

- Refinement can be represented in a generalized framework (such as IOS-specifications)
- Approach can be used for data refinement and behavioral refinement as well
- Approach can be used for proving meta-theoretic properties of refinements (reflexive?, transitive?, composable?) too
- Approach can be used for automated proof support.

References

- [Abr96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
- [A.W97] A.W.Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.
- [BRW03] Achim D. Brucker, Frank Rittinger, and Burkhart Wolff. Hol-z 2.0: A proof environment for z-specifications. *Journal of Universal Computer Science*, 9(2):152–172, February 2003.
- [Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.
- [WD96] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall International Series in Computer Science. Prentice Hall, 1996.