

Deductive Search for Errors in Free Data Type Specifications using Model Generation

Wolfgang Ahrendt

Chalmers University of Technology

Göteborg

CADE-18

Copenhagen, July 2002

Subject

frame: free data type specifications

goal: support for detecting errors in specifications

reveal: non-consequence between specification and conjecture:

$$\text{SPEC} \not\equiv \varphi$$

means: — model construction

— using theory specific calculus

— proof procedure: “model generation” (MGTP)

Algebraic Specification of Abstract Data Types: Syntactical Variations

equality logic: atoms are equations (i.g. no predicates)

variants:

- pure equality (no negation or disjunction)
- Horn equality
- quantifier free (implicit universal closure)
- **full first-order** equality logic

Algebraic Specification of Abstract Data Types: Semantical Variations

unlike usual FOL semantics:
only certain models/domains considered

1. initial semantics:

domain $\hat{=}$ *the* set of 'minimal' equivalence classes over *all* terms

2. constructor generated semantics:

(a) loose semantics:

domain $\hat{=}$ *any* set of equivalence classes over *constructor* terms

functions $\hat{=}$ *any* mappings over the domain

\Rightarrow *non-monomorphic specifications* (different models)

(b) constructive specifications:

monomorphic by syntactical restrictions

(used in automated induction community)

Loose versus Initial Approach

“In software development, design specifications and prototyping by executable specifications are supported by the initial approach; in the loose approach the aim is to cover the whole software development process including requirement specifications.” [M. Wirsing]

algebraic specification languages:

- contemporary quasi standard: CASL
- CASL allows mixing both styles
- ‘loose’ is default

this work: loose specification

monomorphic vs. non-monomorphic

monomorphic approaches (initial semantics, constructive specifications):

- specification has *one* model at most
- $\text{SPEC} \not\models \varphi \iff \text{SPEC} \models \text{Contr}(\varphi)$
 where $\text{Contr}(\varphi) \equiv \neg \text{Cl}_{\forall}(\varphi) \equiv \text{Cl}_{\exists}(\neg\varphi)$
- consistency and provability closely related

non-monomorphic approaches (loose semantics):

- specification can have *several* models
- possible: $\text{SPEC} \not\models \varphi$ and $\text{SPEC} \not\models \text{Contr}(\varphi)$ (under-specification)

Example: Data Type NatStack

```

spec = NatStack

sorts      Nat ::= 0 | s(Nat);
              Stack ::= nil | push(Nat, Stack);

functions  top : Stack → Nat;
              pop : Stack → Stack;
              del : Nat × Stack → Stack;

axioms     top(push(n, st)) ≐ n;
              pop(push(n, st)) ≐ st;
              del(n, push(n, st)) ≐ st;
              n ≠ n' →
                del(n, push(n', st)) ≐ push(n', del(n, st));
              del(n, nil) ≐ nil;

end

```

$\text{NatStack} \not\models \text{pop}(st) \neq st$ *and* $\text{NatStack} \not\models \exists st. \text{pop}(st) \doteq st$

Free Data Types

this work: special case of loose specification: *free* data types only

- *free* $:\iff$ different constructor terms are unequal
- domain is fixed: $\{C_s \mid s \in S\}$ (C_s 'constructor terms of sort s ')
 - different models only vary in:
 interpretation of (non-constructor) function symbols:
 $\mathcal{I}(f) : CT_{s_1} \times \dots \times CT_{s_n} \rightarrow CT_s$
 $val_{\mathcal{I},\beta}(f(t_1, \dots, t_n)) = \mathcal{I}(f)(val_{\mathcal{I},\beta}(t_1), \dots, val_{\mathcal{I},\beta}(t_n))$ (f a function)
 $val_{\mathcal{I},\beta}(c(t_1, \dots, t_n)) = c(val_{\mathcal{I},\beta}(t_1), \dots, val_{\mathcal{I},\beta}(t_n))$ (c a constructor)
- search for models $\hat{=}$ *search for interpretations*

Non-Consequence and Counter Specifications

given:

- specification $\langle \Sigma, AX \rangle$ (with signature Σ and axioms AX)
- conjecture φ

then:

$$\begin{array}{c} \langle \Sigma, AX \rangle \not\models \varphi \\ \iff \\ \langle \Sigma, AX \cup \text{Contr}(\varphi) \rangle \text{ has a model} \end{array}$$

$\langle \Sigma, AX \cup \text{Contr}(\varphi) \rangle$: “counter specification”

Normalization

1. Skolemize away \exists quantifiers (e.g. introduced by *Contr*)
2. add Skolem functions to the *functions* of Σ (not to the constructors)
result: Σ'
3. compute CNF:
result: AX'

$$\langle \Sigma, AX \rangle \not\models \varphi$$

$$\iff$$

normalize($\langle \Sigma, AX \cup \text{Contr}(\varphi) \rangle$) has a model

$$\iff$$

$\langle \Sigma', AX' \rangle$ has a model (with AX' in CNF)

Model Construction = Interpretation Construction

- models are characterized by interpretations
- \Rightarrow deductive method for interpretation construction
- give **operational meaning to axioms and sort declarations**
 1. use axioms to *construct* the interpretation
 2. use sort declarations to *search* for unknown elements in the interpretation
- ‘operational meaning’ given by rules in a theory specific calculus
 - \Rightarrow **transformation** of axioms and signature to rules

Interpretation Representation: Meta Atoms

imagine:
interpretations
are tables

	$\mathcal{I}(f)$
$\langle ct_{11}, \dots, ct_{1n} \rangle$	ct_{10}
$\langle ct_{21}, \dots, ct_{2n} \rangle$	ct_{20}
\vdots	\vdots
\vdots	\vdots
$\langle ct_{i1}, \dots, ct_{in} \rangle$	ct_{i0}
\vdots	\vdots

ct_{ij} :
constructor terms

meta atom: $\mathbf{I}(f, \langle ct_{i1}, \dots, ct_{in} \rangle, ct_{i0})$ represents line



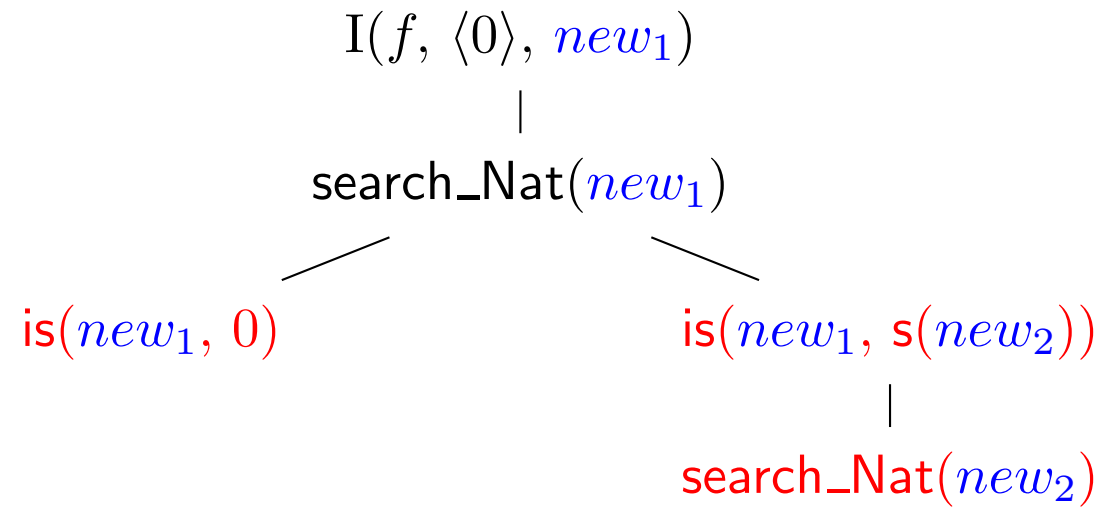
during construction, meta atoms also contain *place holders*

other meta atoms: control *search* for (place holder) replacements

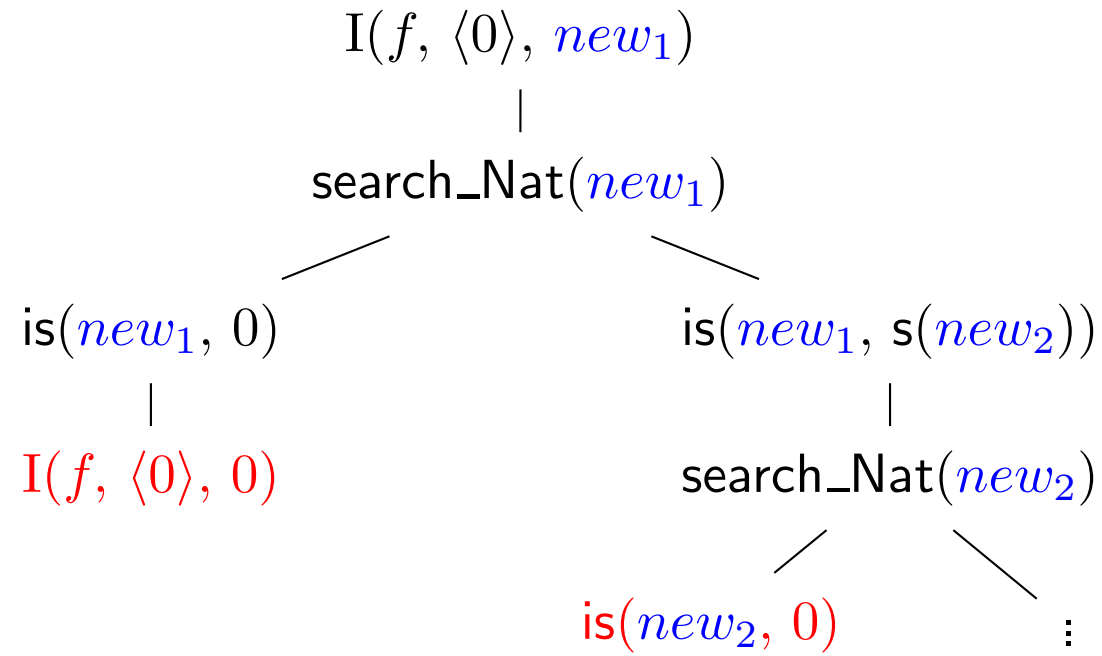
Search for $\mathcal{I}(f)(0)$

$$\begin{array}{c} \mathcal{I}(f, \langle 0 \rangle, new_1) \\ | \\ \text{search_Nat}(new_1) \end{array}$$

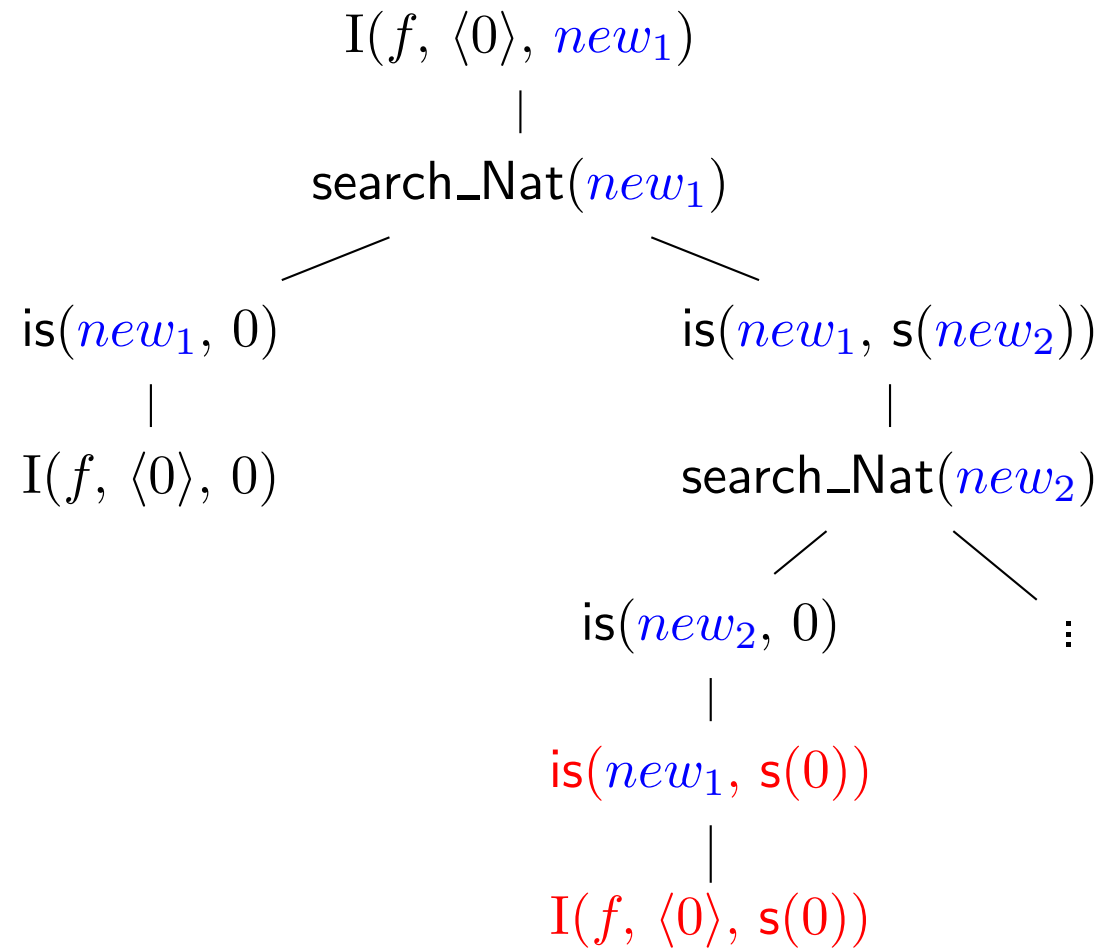
Search for $\mathcal{I}(f)(0)$



Search for $\mathcal{I}(f)(0)$



Search for $\mathcal{I}(f)(0)$



Transformation of Signature

sorts Nat ::= 0 | s(Nat);
 Stack ::= nil | push(Nat, Stack);

↓
transformation
 (simplif. for presentation)

$$\begin{array}{c}
 \text{search_Nat}(x) \\
 \hline
 \text{is}(x, 0) \quad \text{is}(x, \text{s}(new_1)) \\
 \text{search_Nat}(new_1) \\
 + \\
 \text{search_Stack}(x) \\
 \hline
 \text{is}(x, \text{nil}) \quad \text{is}(x, \text{push}(new_1, new_2)) \\
 \text{search_Nat}(new_1) \\
 \text{search_Stack}(new_2)
 \end{array}$$

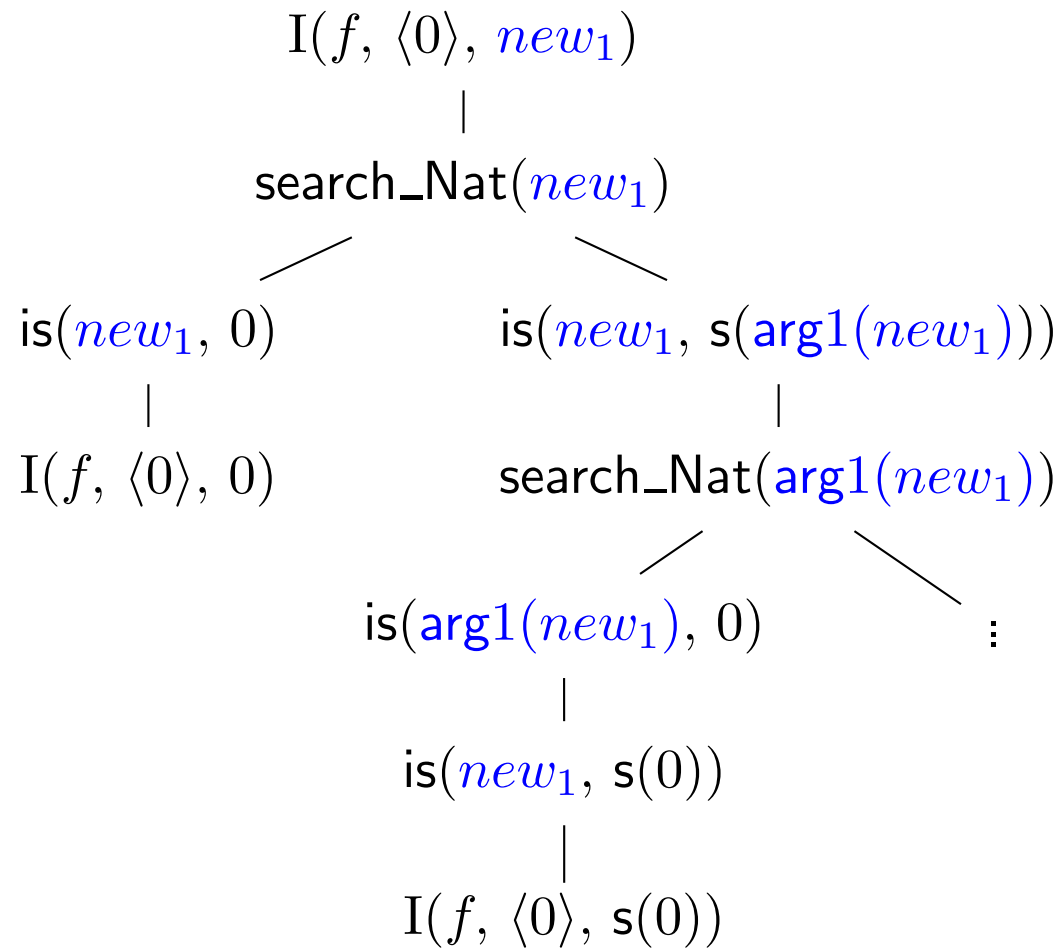
Transformation of Signature

sorts Nat ::= 0 | s(Nat);
 Stack ::= nil | push(Nat, Stack);

↓
transformation
 (not simplif.)

$$\begin{array}{c}
 \text{search_Nat}(x) \\
 \hline
 \text{is}(x, 0) \quad \text{is}(x, \text{s}(\text{arg1}(x))) \\
 \text{search_Nat}(\text{arg1}(x)) \\
 + \\
 \text{search_Stack}(x) \\
 \hline
 \text{is}(x, \text{nil}) \quad \text{is}(x, \text{push}(\text{arg1}(x), \text{arg2}(x))) \\
 \text{search_Nat}(\text{arg1}(x)) \\
 \text{search_Stack}(\text{arg2}(x))
 \end{array}$$

Search for $\mathcal{I}(f)(0)$

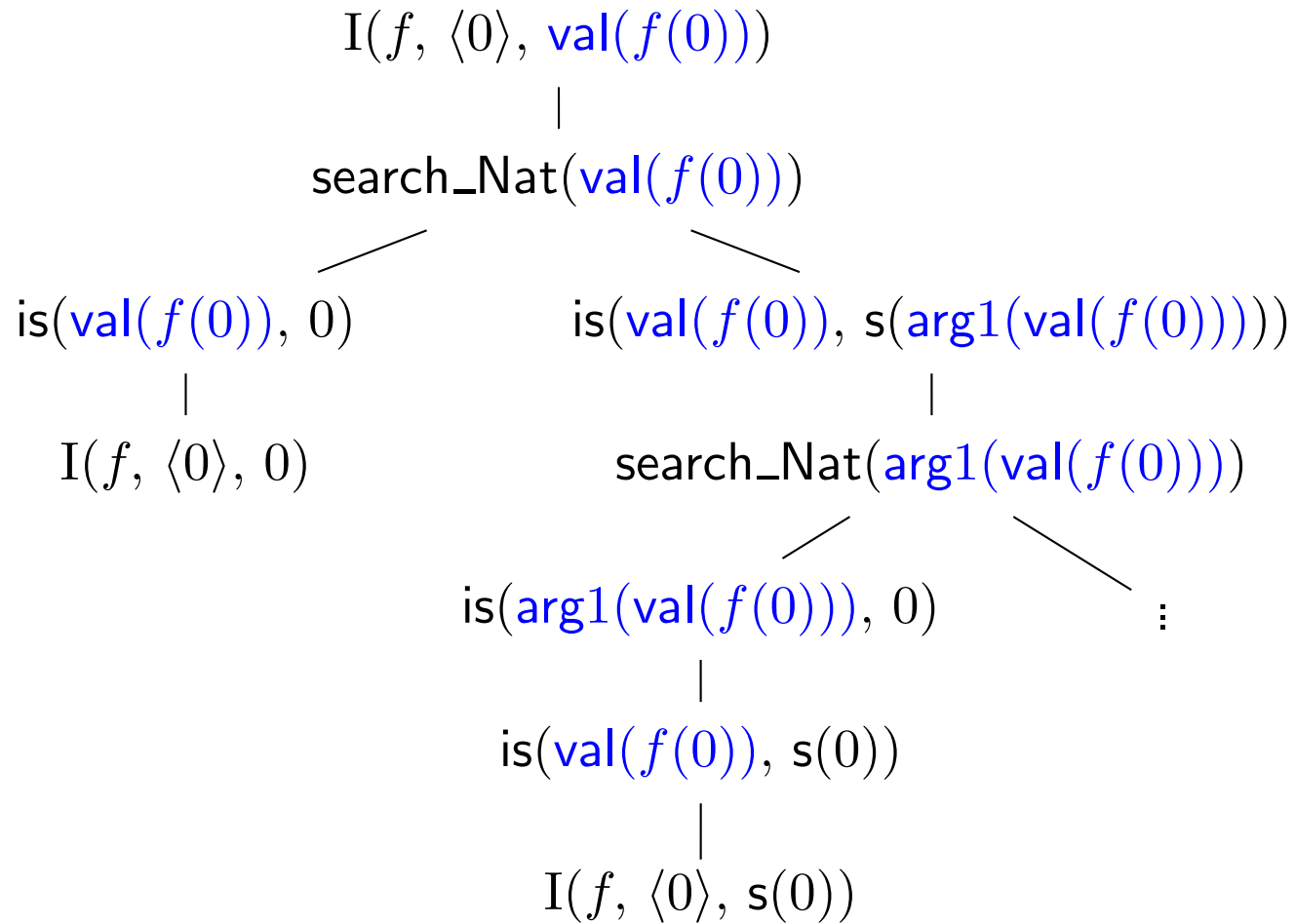


what really is new_1

- What do we know about new_1 in $I(f, \langle 0 \rangle, new_1)$?
- new_1 is equal to $val_{\mathcal{I}}(f(0))$
- \Rightarrow represented by $val(f(0))$

'val' and 'arg' terms are meta terms, talking about valuations and subterms

Search for $\mathcal{I}(f)(0)$



Transformation of Axioms: Example A

axiom $\text{pop}(\text{push}(n, st)) \doteq st;$



transformation

$$\frac{\text{Nat}(n) \quad \text{Stack}(st)}{\text{I}(\text{pop}, \langle \text{push}(n, st) \rangle, st)}$$

Transformation of Axioms: Example B

axiom $n \doteq n' \vee \text{del}(n, \text{push}(n', st)) \doteq \text{push}(n', \text{del}(n, st));$

\downarrow
transformation
 (simplif. for presentation)

	$\text{Nat}(n)$ $\text{Nat}(n')$ $\text{Stack}(st)$
	$\text{I}(\text{del}, \langle n, \text{push}(n', st) \rangle, new_1)$ $\text{is}(new_1, \text{push}(n', new_2))$ $\text{I}(\text{del}, \langle n, st \rangle, new_2)$ $\text{search_Stack}(new_2)$

Transformation of Axioms: Example B

axiom $n \doteq n' \vee \text{del}(n, \text{push}(n', st)) \doteq \text{push}(n', \text{del}(n, st));$

\downarrow
transformation
 (not simplif.)

Nat(n)
 Nat(n')
 Stack(st)

same(n, n')	$I(\text{del}, \langle n, \text{push}(n', st) \rangle, \text{val}(\text{push}(n', \text{del}(n, st))))$ $\text{is}(\text{val}(\text{push}(n', \text{del}(n, st))), \text{push}(n', \text{val}(\text{del}(n, st))))$ $I(\text{del}, \langle n, st \rangle, \text{val}(\text{del}(n, st)))$ search_Stack($\text{val}(\text{del}(n, st))$)
-----------------	--

More Rules (Examples)

functionality of the I predicate:

$$\frac{I(fv, tv, z) \quad I(fv, tv, z')}{\text{same}(z, z')}$$

rules for same:

$$\frac{\text{same}(\text{push}(x_1, x_2), \text{nil})}{*}$$

(rejection)

$$\frac{\text{same}(\text{push}(x_1, x_2), \text{push}(y_1, y_2))}{\text{same}(x_1, y_1) \quad \text{same}(x_2, y_2)}$$

rules are theory- (i.e. specification-) specific:

result of *TransSpec*($\langle \Sigma, AX \rangle$)

Rules as Range Restricted Clauses

$$\begin{array}{c}
 at_1 \\
 \vdots \\
 at_n \\
 \hline
 at_{11} \quad \dots \quad at_{m1} \\
 \vdots \quad \dots \quad \vdots \\
 at_{1n_1} \quad \dots \quad at_{mn_m}
 \end{array}$$

is actually represented as:

$$at_1, \dots, at_n \rightarrow at_{11}, \dots, at_{1n_1} ; \dots ; at_{m1}, \dots, at_{mn_m} .$$

for rejecting rules:

$$at_1, \dots, at_n \rightarrow .$$

r.r. clauses: *input format for variant of positive hyper tableaux,*
called **model generation**

\Rightarrow rules executed by tool **MGTP**

Approximating the Specification

recursive data types: infinite domains

⇒ constructing interpretations in finite time: *program synthesis*

⇒ not feasible automatically (for *loose* specifications)

realizing *automated* search for errors demands concessions:

- **instantiation of variables** in CNF axioms
with **constructor terms** of maximal size n : $\langle \Sigma, AX_{\leq n} \rangle$
- construction of models, i.e. interpretations, for $\langle \Sigma, AX_{\leq n} \rangle$
- user has to decide, if interpretation is extendible to SPEC
help: variation of n

Axiom Instantiation

consider again:

$$\frac{\text{Nat}(m) \quad \text{Stack}(st)}{\text{I}(\text{pop}, \langle \text{push}(m, st) \rangle, st)}$$

- rule application requires $\text{Nat}(t)$ and $\text{Stack}(t')$ on the branch
- \Rightarrow additional rules initialize the branch with
 - $\text{Nat}(t)$ for all $t \in CT_{\text{Nat}}$ with $|t| \leq n$
 - $\text{Stack}(t)$ for all $t \in CT_{\text{Stack}}$ with $|t| \leq n$
 called “ n -initialization”
- total rule set then *constructs model for* $\langle \Sigma, AX_{\leq n} \rangle$

n -restricted Model Correctness and Model Completeness

n -restricted model correctness:

If n -initialised model generation procedure on $\text{TransSpec}(\langle \Sigma, AX \rangle)$ *terminates by saturation*, then

- (a) $\langle \Sigma, AX_{\leq n} \rangle$ *has a model*,
- (b) each interpretation \mathcal{I} corresponding to the I -atoms on the saturated branch characterizes a model of $\langle \Sigma, AX_{\leq n} \rangle$.

n -restricted model completeness:

If $\langle \Sigma, AX_{\leq n} \rangle$ *has a model*, then

- (a) n -initialised fair model generation procedure on $\text{TransSpec}(\langle \Sigma, AX \rangle)$ *terminates by saturation*,
- (b) each interpretation \mathcal{I} corresponding to the I -atoms on the saturated branch characterizes a model of $\langle \Sigma, AX_{\leq n} \rangle$.

Unrestricted Model Completeness

corollary:

unrestricted model completeness:

$\langle \Sigma, AX \rangle$ has a model, then *for each* n , the n -initialised fair model generation procedure on $\text{TransSpec}(\langle \Sigma, AX \rangle)$ terminates by saturation.

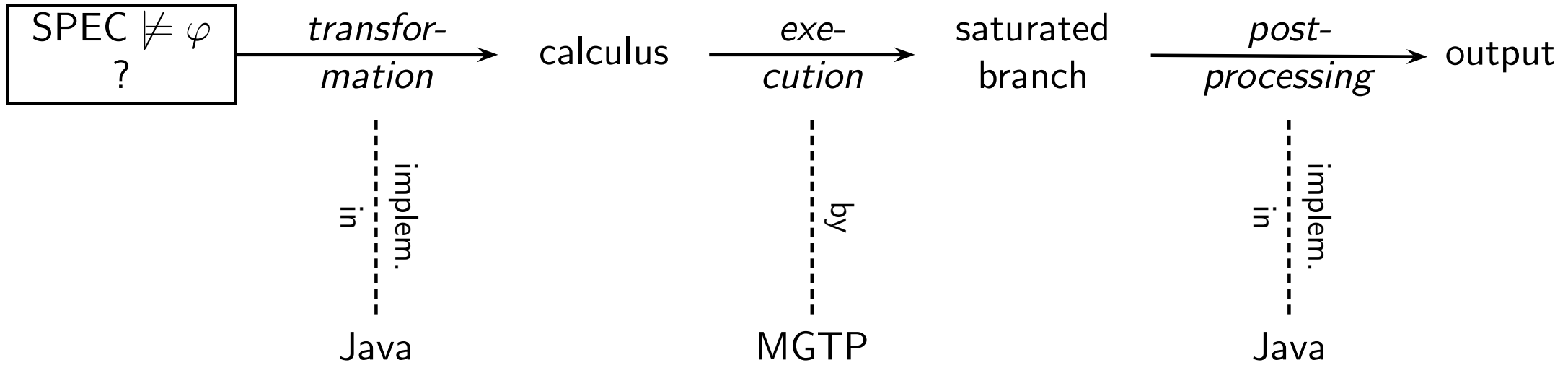
reasons:

- constructor generatedness
 $\Rightarrow \langle \Sigma, AX \rangle$ equivalent to $\langle \Sigma, AX_{\leq \infty} \rangle$
- the logic is *monotonous*
 \Rightarrow a model of $\langle \Sigma, AX_{\leq \infty} \rangle$ is also a model of $\langle \Sigma, AX_{\leq n} \rangle$

due to limiting *instantiations of axioms*

no similar result when working with limited *domain size*

Realization



MGTP = 'Model Generation Theorem Prover' (Univ. Fukuoka)

Example: NatStack

spec = NatStack

sorts Nat ::= 0 | s(Nat);
 Stack ::= nil | push(Nat, Stack);

functions top : Stack \rightarrow Nat;
 pop : Stack \rightarrow Stack;
 del : Nat \times Stack \rightarrow Stack;

axioms top(push(n , st)) \doteq n ;
 pop(push(n , st)) \doteq st ;
 del(n , push(n , st)) \doteq st ;
 $n \neq n' \rightarrow$
 del(n , push(n' , st)) \doteq push(n' , del(n , st));
 del(n , nil) \doteq nil;

end

$$\text{del}(\text{top}(st), st) \doteq \text{pop}(st)$$

Example Run

system input:

$$\text{del}(\text{top}(st), st) \doteq \text{pop}(st) ?$$

system output (the given limit is 4):

the conjecture

$$\text{del}(\text{top}(ST), ST) = \text{pop}(ST)$$

is **violated** by the following variable assignment:

ST : nil

and by the following evaluation of subterms:

$\text{del}(\text{top}(ST), ST) : \text{nil}$

$\text{top}(ST) : 0$

$\text{pop}(ST) : \text{push}(0, \text{nil})$

The interpretation found by the system satisfies the axioms,
if instantiated by constructor terms with less than 4 constructors!

MergeSort

```
sort(empty) ≐ empty;  
sort(append(l, l')) ≐ merge(sort(l), sort(l'));  
  
append(empty, l) ≐ l;  
append(cons(n, l), l') ≐ cons(n, append(l, l'));  
  
merge(l, empty) ≐ l;  
merge(empty, l') ≐ l';  
less(n, n') ≐ tt →  
    merge(cons(n, l), cons(n', l')) ≐ cons(n, merge(l, cons(n', l')));  
less(n, n') ≐ ff →  
    merge(cons(n, l), cons(n', l')) ≐ cons(n', merge(cons(n, l), l'));  
⋮
```

[Reif, Schellhorn, Thums: Ulmer Informatik-Berichte 2000-06]

MergeSort

```

sort(empty) ≐ empty;
sort(append(l, l')) ≐ merge(sort(l), sort(l'));

append(empty, l) ≐ l;
append(cons(n, l), l') ≐ cons(n, append(l, l'));

merge(l, empty) ≐ l;
merge(empty, l') ≐ l';

less(n, n') ≐ tt →
    merge(cons(n, l), cons(n', l')) ≐ cons(n, merge(l, cons(n', l')));
less(n, n') ≐ ff →
    merge(cons(n, l), cons(n', l')) ≐ cons(n', merge(cons(n, l), l'));
⋮

```

[Reif, Schellhorn, Thums: Ulmer Informatik-Berichte 2000-06]

Conjecture:

sort(cons(n, empty)) ≐ cons(n, empty) ?

Example Run on MergeSort

system input:

$\text{sort}(\text{cons}(n, \text{empty})) \doteq \text{cons}(n, \text{empty})$?

system output (the given limit is 2):

the conjecture

$\text{sort}(\text{cons}(N, \text{empty})) = \text{cons}(N, \text{empty})$

is violated by the following variable assignment:

$N : \text{zero}$

and by the following evaluation of conjecture subterms:

$\text{sort}(\text{cons}(N, \text{empty})) : \text{empty}$

$\text{cons}(N, \text{empty}) : \text{cons}(\text{zero}, \text{empty})$

Future Work

- further optimizations in the transformation
- axiom reduction
- parameter types (e.g. `Stack(Elem)`)
- distinguish under-specified and completely specified functions
- examine: preprocessing + propositional sat-solving

Related Areas

1. model construction

- many works in the fields $\left\{ \begin{array}{l} \text{FOL} \\ \text{finite models} \end{array} \right.$
- recursive, constructor generated data types not expressible there

2. non-consequence and consistency

- many works in the fields $\left\{ \begin{array}{l} \text{initial (or rewrite) semantics} \\ \text{constructive specifications} \end{array} \right.$
- [Reif,Schellhorn,Thums01]: loose specifications (free and non-free)

[Reif,Schellhorn,Thums01]:

- counter example $\hat{=}$ variable assignment only
- as here: total correctness of ‘counter example’ not guaranteed
- no notion of ‘restricted’ correctness, no completeness result
- no model or interpretation to ‘narrow’ under-specification

Summary

- method tailor-made for *detecting non-consequence* between free data type specifications and conjectures
 - termination by limiting the *instantiation* of axioms
 - price: *restricted* model correctness
 - no 'domain size limits'
 - gain: *un-restricted* model completeness
- ⇒ the system detects too many non-consequences, but it detects all
- user decision based on:
examining output interpretation + varying instance limit

Example: Data Type p_Forever

```
spec = p_Forever
```

```
  sorts      Nat ::= 0 | s(Nat);
            Bool ::= tt | ff;
```

```
  functions  p : Nat → Bool;
```

```
  axioms     p(0) ≐ tt;
            p(x) ≐ tt → p(s(x)) ≐ tt;
```

```
end
```

$$\models p(y) \doteq \text{tt}$$

but:

the conjecture

$p(y) = \text{tt}$

is **violated** by the following variable assignment:

$y : s(s(s(0)))$

$y : s(s(s(s(0))))$

$y : s(s(s(s(s(0)))))$