An Overview of Recent Research Activities around CafeOBJ

FUTATSUGI,Kokichi
二木 厚吉

JAIST
(Japan Advanced Institute of Science and Technology)

CafeOBJ

Topics of this talk

- Some introductory remarks on Formal Methods, CafeOBJ, and Proof Scores
- Combination of inference and search in the proof score method
  - Abstraction-with-Inference + search
  - Induction Guided Falsification
    - Backward-search (inference) + forward-search (search)
- Sound and complete proof rules underlie the proof score method
- Concluding remarks

CafeOBJ

Our Perception on Formal Methods and Specification Verifications

CafeOBJ

Application areas of formal methods (FM)

1. Analysis and verification of developed program codes (post-coding)

2. Analysis and verification of (models/specs of) domains, requirements, and designs before/ without coding (pre-coding or without coding)

Successful application of formal methods to the area of (modeling/specification of) domains, requirements, designs can bring drastic good effects for systems developments, but it is not well exploited and/or practiced yet.

specification = description of model

CafeOBJ

The current situation of FM

- Verification with formal specifications still have a potential to improve the practices in upstream (pre-coding) of systems development processes
- Model checking has brought a big success but still has limitations
  - It is basically "model checking" for program codes
    - Still mainly for post-coding
  - Infinite state to finite state transformation can be unnatural and difficult
- Established interactive theorem provers (Isabelle/HOL, Coq, PVS, etc.) are still to be well accepted to ordinary software/systems engineers
  - especially in upstream (pre-coding) phase

CafeOBJ

Our approach

- Reasonable blend of user and machine capabilities, intuition and rigor, high-level planning and tedious formal calculation
  - fully automated proofs/verifications are not necessary good for human beings to perceive logical structures of real problems/systems
  - interactive understanding/description of real problem domains/requirements/designs is necessary

⬇

Proof Score Approach

CafeOBJ

Proof Score
as a Complete Set of Symbolic Test cases

- Domain/requirement/design engineers are expected to construct proof scores together with formal specifications
- Proof score is a complete set of symbolic test cases such that when executed (or evaluated/ reduced) and everything evaluates as expected, then the desired property is convinced (or proved) to hold. Proof score is supposed to be read by engineers.
  - Proof by construction/development
  - Proof by computation/reduction/rewriting
  - Test Driven (Specification) Development

Development of proof scores in CafeOBJ

- Many simple proof scores are written in OBJ language from 1980's; some of them are not trivial
- From around 1997 CafeOBJ group at JAIST use proof scores seriously for verifying specifications for various examples
  - From static to dynamic/reactive system
  - From ad hoc to more systematic proof scores
  - Introduction of OTS (Observational Transition System) was a most an important step

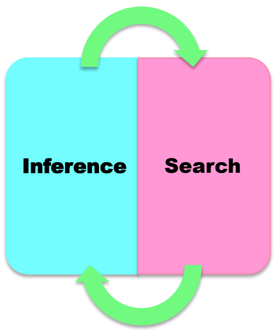
Some achievements of CafeOBJ/OTS proof score approach

CafeOBJ/OTS approach has been applied to the following kinds of problems and found usable:

- Some classical mutual exclusion algorithms
- Some real time algorithms
  e.g. Fischer's mutual exclusion protocol
- Railway signaling systems
- Authentication protocol
  e.g. NSLPK, Otway-Rees, STS protocols
- Practical sized e-commerce protocol of SET
  (some of proof score exceeds 60,000 lines;
  specification is about 2,000 lines,
  20-30 minutes for reduction of the proof score)
- UML semantics (class diagram + OCL-assertions)
- Formal Fault Tree Analysis
- Secure workflow models, internal control

CafeOBJ

Verification by Inference and Search in Proof Scores



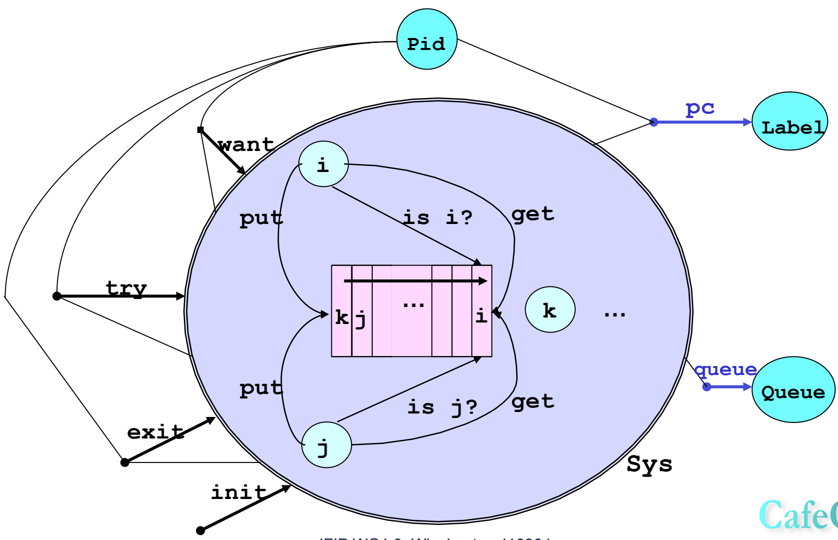
CafeOBJ

Two topics

- Abstraction by inference (TP) and counter example finding by search (MC)
  - QLOCK example
- Counter example finding by MC (Search) and TP (Inference)
  - NSPK example

CafeOBJ

Modeling QLOCK (via Signature Diagram) with OTS (Observational Transition System)



CafeOBJ

CafeOBJ signature for QLOCKwithOTS

```
-- state space of the system
*[Sys]*                                        system sort declaration

-- visible sorts for observation
[Queue Pid Label]                              visible sort declaration

-- observations
bop pc : Sys Pid -> Label                      observation declaration
bop queue : Sys -> Queue

-- any initial state
bop init : -> Sys {constr}                     initial state declaration
-- actions
bop want : Sys Pid -> Sys {constr}
bop try  : Sys Pid -> Sys {constr}             action declaration
bop exit : Sys Pid -> Sys {constr}
```

CafeOBJ

Transition system for QLOCK (1)

```
mod* QLOCKconfig {
  inc(QLOCK)
  [ Config ]
  op <_> : Sys -> Config .
}
-- pre-transiton system with an agent/process p
mod* QLOCKpTrans {
  inc(QLOCKconfig)
  op p : -> PidConst .
  var S : Sys .
  -- possible transitions
  ctrans < S > => < want(S,p) > if c-want(S,p) .
  ctrans < S > => < try(S,p) >  if c-try(S,p) .
  ctrans < S > => < exit(S,p) > if c-exit(S,p) .
}
```

CafeOBJ

Transition system for QLOCK (2)

```
-- transition system with 2 agents i j
mod* QLOCKijTrans {
  inc((QLOCKpTrans * {op p -> i}) +
      (QLOCKpTrans * {op p -> j}))
}

-- transition system with of 3 agents i j k
mod* QLOCKijkTrans {
  inc(QLOCKijTrans +
      (QLOCKpTrans * {op p -> k}))
}
```

CafeOBJ

Search predicate of CafeOBJ a la Maude's search command

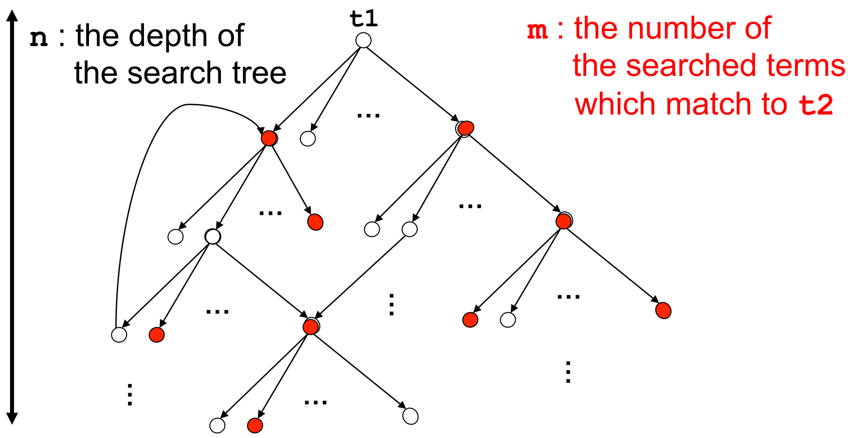
CafeOBJ System has the following built-in predicate:
- **Any** is any sort (that is, the command is available for any sort)
- **NzNat*** is a built-in sort containing non-zero natural number and the special symbol "*" which stands for infinity

```
pred _=(_,_)=>*_ : Any NzNat* NzNat* Any
```

`(t1 =(m,n)=>* t2)` returns **true** if `t1` can be translated (or rewritten), via more than 0 times transitions, to some term which matches to `t2`. Otherwise, it returns **false**. Possible transitions/rewritings are searched in breadth first fashion. `n` is upper bound of the depth of the search, and `m` is upper bound of the number of terms which match to `t2`. If either of the depth of the search or the number of the matched terms reaches to the upper bound, the search stops.

CafeOBJ

t1 =(m,n)=>* t2



CafeOBJ

suchThat predicate

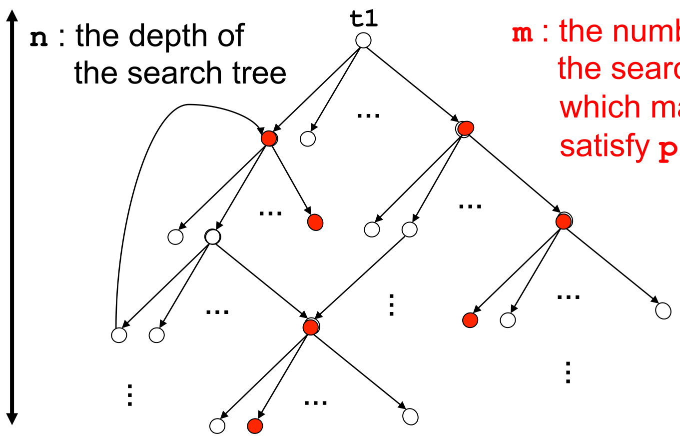
`t1 =(m,n)=>* t2 suchThat pred1(t2)`

pred1(t2) is a predicate about **t2** and can refer to the variables which appear in **t2**.
pred1(t2) enhances the condition used to determine the term which matches to **t2**.

CafeOBJ

t1 =(m,n)=>* t2 suchThat pred1(t2)

n : the depth of the search tree



m : the number of the searched terms which match to **t2** and satisfy **pred(t2)**

`withStateEq` predicate

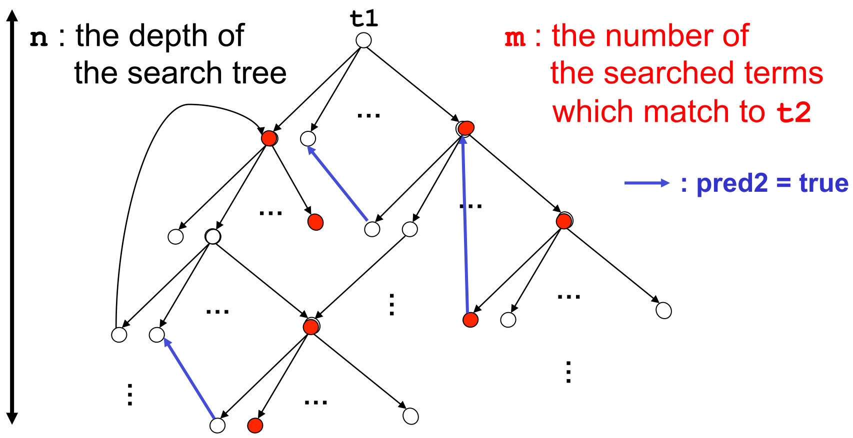
```
t1 =(m,n)=>* t2
   withStateEq pred2(V1:St,V2:St)
```

Pred2(V1:St,V2:St) is a binary predicate of two arguments with the same sort `St` of the term **t2**.
Pred2(V1:St,V2:St) is used to determine a newly searched term (a state configuration) is already searched one.
If this **withStateEq** predicate is not given, the term identity binary predicate is used for the purpose.

Using both of `suchTant` and `withStateEq` is also possible

```
t1 =(m,n)=>* t2 suchThat pred1(t2)
             withStateEq pred2(S1:Sort,S2:Sort)
```

t1 =(m,n)=>* t2
withStateEq pred2(V1:St,V2:St)



CafeOBJ

Verification by Searching with Observational Equivalence

```
red in (QLOCKijTrans + QLOCKobEq + MEX) :
       < init > =(*,*)=>* < S:Sys >
       suchThat (not mutualEx(S,i,j))
       withStateEq (C1:Config =ob= C2:Config) .
```

This CafeOBJ code searches for a counter example of mutual exclusion property in the whole state space Sys(i,j) of two agents system. If this returns `false`, the two agents system is verified to have the mutual exclusion property.

CafeOBJ

Simulation of any number of agents systems by the two agents system

Proof scores of simOfQLOCKbyQLOCKijPS.mod csQtopPS.mod

verify the following

let i and j be any two distinctive process identifiers, and let Sys(i,j) be the state space of QLOCK with only the two processes i and j,
then
(there is a counter example in Sys)
implies
(there exits a counter example in Sys(i,j))
that is,
(for-all t:Sys(i,j)).pred(t,i,j)
implies
(for-all s:Sys).pred(s,i,j)

CafeOBJ

Counter example finding by forward and backward search -- another kind of collaborative use of MC & TP

- MC & TP can be collaboratively used to find a counterexample that exists at a deep position.
  - Properties concerned are *invariants*.
  - *Bounded model checking* (*BMC*) is used as an MC technique.
  - *Induction* is mainly used as a TP technique.
- We have proposed a collaborative use of BMC & induction to find a deep counterexample for invariants: *Induction-Guided Falsification* (*IGF*).

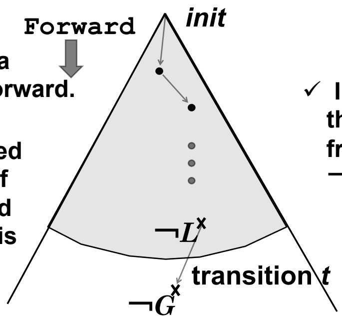
K. Ogata, M. Nakano, W. Kong, K. Futatsugi: Induction-Guided Falsification, 8th ICFEM, LNCS 4260, Springer, pp.114-131 (2006).

CafeOBJ

Induction-Guided Falsification (IGF)

- ✓ Suppose that a counterexample of an invariant *G* exists outside of the bounded reachable state space that can be exhaustively traversed.
- ✓ induction may conjecture a lemma *L* such that its counterexample exists in the space.

- ✓ BMC tries to find a counterexample forward.
- ✓ IGF can be regarded as a combination of forward & backward reachability analysis methods.



- ✓ Induction tries to show that there are no paths from any states such that ¬*G* to any initial states.

Backward

CafeOBJ

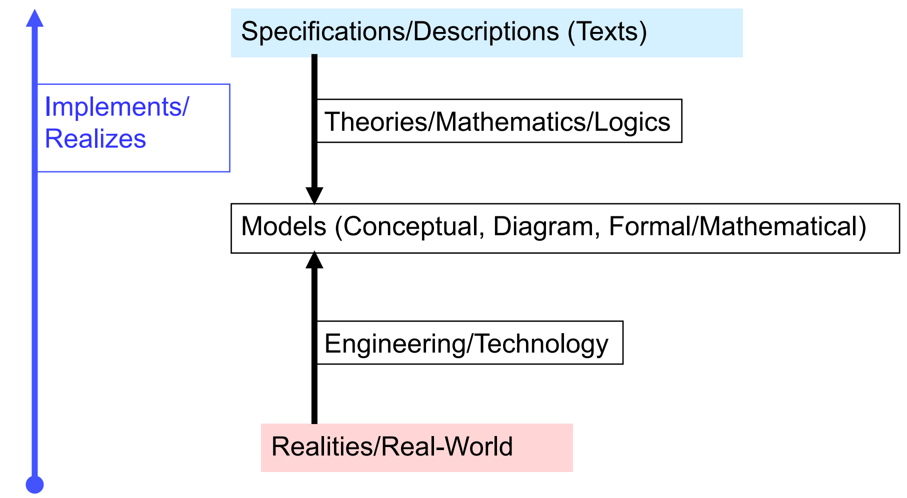
Sound and “Complete” Proof Rules for Proof Scores

CafeOBJ

Topics

- Specification/Descriptions, Models, and Realities
- Constructor-based Order Sorted Algebra
- Satisfaction of a Property by a Specification
  - SPEC |= prop
- Proof rules for SPEC |= prop and SPEC |- prop

Specifications, Models, Realities



Specification

An **constructor-based equational specification SPEC** in CafeOBJ (a text in the CafeOBJ language with only equational axioms) is defined as a pair **(Sig,E)** of order-sorted constructor-based signature **Sig** and a set **E** of conditional equations over **Sig**. A signature **Sig** is defined as a triple **(S,F,F^c)** of an partially ordered set **S** of sorts, an indexed family **F** of sets of **S**-sorted functions/operations, and a set **F^c** of constructors. **F^c** is a family of subsets of **F**, i.e. $\mathbf{F^c} \subseteq \mathbf{F}$.

SPEC = ((S,F,F^c),E)

CafeOBJ

Model: (S,F)-Algebra

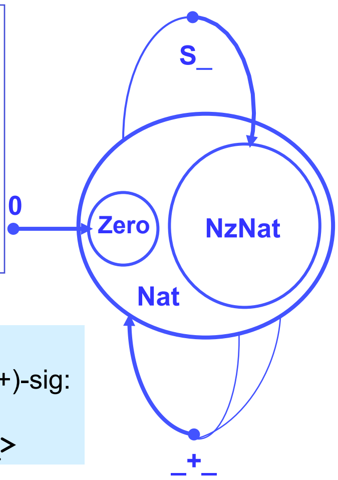
A formal/mathematical **model** of a specification **SPEC = ((S,F,F^c),E)** is an reachable order-sorted **algebra A** which has the signature **(S,F)** and satisfies all equations in **E**.

An order-sorted algebra which has a signature (S,F) is called an **(S,F)-algebra.** An **(S,F)-algebra A** interprets a sort symbol s in **S** as a (non empty) set $\mathbf{A_s}$ and an operation (function) symbol **f :s1 s2 …sn->s(n+1)** in **F** as a function $\mathbf{A_f : A_{s1},A_{s2},..,A_{sn} \rightarrow A_{s(n+1)}}$. The interpretation respects the order-sort constrains.

CafeOBJ

An example of Signature and its Algebra

```
-- Let (PNAT+)-sig be
-- the  signature of PNAT+
-- sort
[ Zero NzNat < Nat ]
-- operators
op 0 :   -> Nat {constr}
op s_ :  Nat -> NzNat {constr}
op _+_ : Nat Nat -> Nat
```



A (PNAT+)-sig-algebra
Order-Sorted Algebra with Signature (PNAT+)-sig:

`<Nat, NzNat, Zero; 0, s_, _+_>`

Model: (S,F,F^C)-Algebra

If a sort s ∈ S is the co-arity of some operator f ∈ F^C, the sort s is called a **constrained sort**. A sort which is not constrained is called a **loose sort**.

An (S,F)-algebra A is called **(S,F,F^C)-algebra** if any value v ∈ A_s for any constrained sort s ∈ S is expressible only using
(1) function A_f for f ∈ F^C
and
(2) function A_g for g ∈ F whose co-arity is loose sort .

(S,F,F^C)-algebra can also be called **F^C-reachable algebra**

Valuation, Evaluation

A **valuation** (or an assignment) is a sort preserving map from the (order-sorted) set of variables of a specification to an order-sorted algebra (a model), and assigns values to all variables.

Given a model **A** and a valuation **v**, a **term t** of sort **s,** which may contain variables, is evaluated to a **value** $\mathbf{A_v(t)}$ in $\mathbf{A_s}$

CafeOBJ

Equation

Given terms t, t', t_1, t_1', t_2, t_2'…t_n, t_n' , a **conditional equation** is a sentence of the form:

$$\mathbf{t = t' \text{ if } (t_1 = t_1') \wedge (t_2 = t_2') \wedge \ldots \wedge (t_n = t_n')}$$

An ordinary equation is a sentence of the form:

t = t'

that is n=0.

A conditional equation in CafeOBJ notation:

t = t' if c

where t,t' are any terms and c is a Boolean term is an abbreviation of

t = t' if c = true

CafeOBJ

Satisfiability of Equation

An ordered-sorted algebra **A** **satisfies** a conditional equation:

$t = t'$ if $(t_1 = t_1')$ /\ $(t_2 = t_2')$ /\…/\ $(t_n = t_n')$

iff

$A_v(t_1)=A_v(t_1')$ and $A_v(t_2)=A_v(t_2')$ and…and $A_v(t_n)=A_v(t_n')$ implies $A_v(t)=A_v(t')$

for any valuation **v** .

The satisfaction of an equation by a model **A** is denoted by
A |= ($t = t'$ if $(t_1 = t_1')$ /\ $(t_2 = t_2')$ /\…/\ $(t_n = t_n')$)

CafeOBJ

CafeOBJ _=_ (meta-level equality) and Boolean _=_ (object-level equality)

If a specification SP includes,

```
op _=_ : S S -> Bool .
eq (X = X) = true .
ceq X = Y if (X = Y) .
```

then

SP |= t=t' if $(t_1=t_1')$/\ $(t_2=t_2')$/\…/\$(t_n=t_n')$

iff

SP |= ((t_1=t_1' and t_2=t_2' and …and t_n=t_n') implies t=t') = true .

1. **Object-level equality can substitute for meta-level equality**
2. **Every sentence (conditional equation) can be written as a Boolean term.**

CafeOBJ

SPEC-algebra

For a specification **SPEC** = ((**S,F,F^c**), **E**), a **SPEC-algebra** is a **(S,F,F^C)-algebra** which satisfies all equations in **E** .

CafeOBJ

Satisfiability of property by specification: SPEC |= prop

A specification **SPEC** = **((S,F,F^c),E)** is defined to satisfy a property **p** (a term of sort **Bool)** iff **A |= (p** = true) holes for any **SPEC-algebra A.**

The satisfaction of a predicate **prop** by a specification **SPEC** = **((S,F,F^c),E)** is denoted by:

SPEC |= p or **E |= p**

A most important purpose of developing a specification **SPEC = ((S,F,F^c),E)** in CafeOBJ is to check whether

SPEC |= prop

holds for a predicate **prop** which describes some important property of the system which **SPEC** specifies.

CafeOBJ

Proof rules for SPEC |= prop (semantic entailment)

For doing formal verification, it is common to think of syntactic (proof theoretic) entailment:

SPEC |- prop

which corresponds to semantic entailment:

SPEC |= prop .

We have developed a sound and *quasi* complete set of proof rules for |- which satisfies:

SPEC |- prop iff SPEC |= prop

for unstructured specifications and constitutes a theoretical foundation for verifications with proof scores.

CafeOBJ

Proof Rules (1) -- entailment system (S, P, or Ei denotes a set of equations)

Monotonicity: $\dfrac{}{E1 \mid\!- E2}$ for any $E2 \subseteq E1$

Transitivity: $\dfrac{E1 \mid\!- E2 ,\ E2 \mid\!- E3}{E1 \mid\!- E3}$

Unions: $\dfrac{E1 \mid\!- E2 ,\ E1 \mid\!- E3}{E1 \mid\!- E2\ U\ E3}$

Translation: $\dfrac{S \mid\!-_{\Sigma} P}{\varphi(S) \mid\!-_{\Sigma'} \varphi(P)}$ for any signature morphism $\varphi: \Sigma \to \Sigma'$

CafeOBJ

Proof Rules (2) -- equational reasoning

(t and ti denotes terms, f denotes operator, p denotes predicate)

Reflexivity: $\dfrac{}{\text{|- \{t=t\}}}$

Symmetry: $\dfrac{}{\text{\{t1=t2\} |- \{t2=t1\}}}$

Transitivity: $\dfrac{}{\text{\{t1=t2, t2=t3\} |- \{t1=t3\}}}$

Congruence: $\dfrac{}{\text{\{t1=t1',t2=t2',...,tn=tn'\} |- f(t1,t2,...,tn)=f(t1',t2',...,tn')}}$

P-Congruence: $\dfrac{}{\text{\{t1=t1',t2=t2',...,tn=tn'\} U \{p(t1,t2,...,tn)\} |- p(t1',t2',...,tn')}}$

CafeOBJ

Proof Rules (3)

(H denotes a set of equations, p denotes predicate, X, Y, or Z denotes set of variables, x denotes variable)

Implication: $\dfrac{\text{S |- P U \{(/\textbackslash H=>p)\}}}{\text{S U H |- P U \{p\}}}$ and $\dfrac{\text{S U H |- P U \{p\}}}{\text{S |- P U \{(/\textbackslash H=>p)\}}}$

Substitutivity: $\dfrac{\text{S |- P}}{\text{S U \{(}\forall\text{x)p\} |- P U \{(}\forall\text{Y)p(x<-t)\}}}$

Generalization: $\dfrac{\text{S } |\text{-}_{\Sigma} \text{ P U \{(}\forall\text{Z)p\}}}{\text{S } |\text{-}_{\Sigma(Z)} \text{ P U \{p\}}}$ and $\dfrac{\text{S } |\text{-}_{\Sigma(Z)} \text{ P U \{p\}}}{\text{S } |\text{-}_{\Sigma} \text{ P U \{(}\forall\text{Z)p\}}}$

CafeOBJ

Proof Rules (4) – these are infinite in nature

(p denotes predicate, Y denotes set of variables, x denotes variable, f denotes a function, ti denotes a term)

C-Abstraction (Constructor Abstraction):

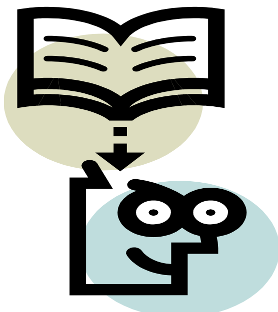
$$\frac{\{\ (S \vdash \{(\forall Y)p(x{<}{-}t)\})\ \mid\ t \text{ is constructor Y-term, Y are loose vars}\ \}}{S \vdash \{(\forall x)p\}}$$

Case Analysis:

$$\frac{\{\ (S \cup \{f(t1,\ldots,tn)=t\} \vdash_{\Sigma(Y)} \{p\})\ \mid\ t \text{ is constructor Y-term, Y are loose vars}\ \}}{S \vdash_{\Sigma} \{p\}}$$

CafeOBJ

Concluding Remarks



CafeOBJ

Three levels of CafeOBJ applications

1. Construct formal models; describe formal specifications
2. Do rapid prototypings or animations and check the properties of specifications; execute specifications for validations/verifications
3. Write proof scores to verify properties of specifications; verifications/proofs with reductions/ rewritings

Choose an appropriate level
depending on problems and situations

CafeOBJ

Prerequisites for proof score writing in CafeOBJ (1)

- Algebraic modeling: development of algebraic specifications
  - defining signature for a real problem
  - expressing the semantics of a problem in equations
    - more exactly, expressing the problem in reduction rules

CafeOBJ

Prerequisites for proof score writing in CafeOBJ (2)

- Equational logic, rewriting, and propositional calculus
  - equationl reasoning
    - equivalence relation, equational calculus, …
  - propositional calculus with “xor” normal forms which has the complete rewriting calculus
  - reduction/rewriting
    - termination, confluence, sufficiently completeness

CafeOBJ

Prerequisites for proof score writing in CafeOBJ (3)

- Proof by induction and case analysis
  - case splitting using constructors or key predicates in specifications
  - discovery of lemmas
  - decomposition of a goal predicate into an appropriate conjunctive form

These are the most difficult parts of proof score writing

But this is common to any kind of interactive verifiers!

CafeOBJ

Traceability in proof score approach with CafeOBJ

- All reductions are done exactly using equations in specifications as rewriting rules
  - this make it easy to detect necessary changes in specs for letting something happen (or not happen)
- Usually reductions are sufficiently fast, and encourage prompt interactions between user and system

This is a quit unique feature of the proof score approach with CafeOBJ comparing to other verification method which often involves several formalisms/logics and translations between them

CafeOBJ

Equational proofs by reduction/rewriting

Why do we care about "equational reasoning by reduction" ?

- It is simple and powerful and a promising light weighted formal reasoning method
  - easy to understand and can be more acceptable for software engineers
- It supports transparent relation between specs and reasoning by reduction (good traceability)

CafeOBJ

Future Issues

- Development of the environment for proof score constructions
  - Standard platforms for programming environment can be naturally used
  - Proof score checker to check correctness of the proof scores as independently as possible
  - Farther development of the Kumo/Tatami scheme to realize a web (or hypertext) based constructions of specs and proof scores
- Serious development of practical domain/requirement/design specifications in the application area like e-government, e-commerce, open standards for automotive software, etc.
  - The development should aim at reasonable balance of informal and the formal specifications, and verify as much as meaningful and important properties of the models/problems the specifications are describing

CafeOBJ

CafeOBJ official home page

http://www.ldl.jaist.ac.jp/cafeobj/

CafeOBJ