Constructing Proof Scores in CafeOBJ

FUTATSUGI, Kokichi
二木厚吉
Japan Advanced Institute of Sciecne and Technology
(JAIST)

at LAC2025
on 30 September 2025

Contents of Talk

(1) Specification Verification, CafeOBJ, Proof Scores

(2) Proof Tree Calculus and Well-Founded Induction (PTcalc+WFI)

(3) Constructing Proof Scores in CafeOBJ

based on the appendix A of the following paper:
"Advances of Proof Scores in CafeOBJ"
http://arxiv.org/abs/2112.10373v3
which is a revised and extended version of
the published journal (Sci.Comp.Prog.) paper
with the same title

(4) Concluding Remarks and Future Perspective

What is Specification Verification?

- Specification is a description of models, and **Specification Verification** is to show that the models have desirable properties by deducing the properties from the specification.
- **Specification Verification** in this sense is theorem proving where the specification is a set of axioms and the desirable properties are theorems.

Theorem Proving vs. Specification Verification

- The primary goal of **Theorem Proving** is to show that theorems can be deduced from axioms with deduction rules. Usually the axioms are already established and does not change.
- The primary goal of **Specification Verification** is to check the quality of a specification (a set of axioms) against the desirable properties. The specification is supposed to be improved/refined to have the desirable properties through specification verification.

Characteristics of Specification Verification

- The improvements of a specification (spec for short) include the **addition of necessary axioms**, the **deletion of unnecessary axioms**, and the **improvement of the spec's module structure**.
- For achieving the improvements, a spec and the machinery of the spec verifivation are better to be clear and transparent. That is, the followings are better to be clear and precisely defined, and hopefully simple and transparent.
  (i) **Models of a spec** (models of a system specified),
  (ii) **inference (deduction) rules** used,
  (iii) **rationale for assumptions used** (i.e. why an assumption is justified to be used or not).

All models are wrong

- **"All models are wrong, some are useful."** and specification verification (i.e. modeling, description, and verification of models) is challenging.
- Specification Verification is **a good (may be the best) way for getting a high-quality specification** (i.e. useful description of models).
- Critical flaws continue to exist at the level of domain, requirement, and/or design specification, and specification verification is still **one of the most important challenges in software/system engineering**.

CafeOBJ (`https://cafeobj.org/`)

- CafeOBJ is an **executable algebraic spec language system** based on **equational logic** and **rewriting execution** with transparent and precise definition of the models and the inference/execution rules.
- CafeOBJ's **powerful module system** makes it possible to describe the assumption/conclusion relations in a clear and transparent way.
- **CafeOBJ** can be used for **specification verification** by writing a **proof score** for verifying that any model of a specification (a CafeOBJ module) M satisfies a property of interest p (a Boolean ground term). In symbols $M \models p$.

A little bit of CafeOBJ history

- 1983-1984: Design/implementation of the OBJ2 language system at SRI International (Standord Research Institute)
- 1985-1995: Several CafeOBJ design and implementation attempts inheriting achievements of the OBJ project at SRI
- 1996-1998: An intensive international CafeOBJ project
  - Six Japanese Companies, Five Japanese Universities, Three Foreign Research Group participate
- 1999-2000: Sufficiently reliable and usable CafeOBJ system was available
- 2000-2021: Researches on **Specification Verification in CafeOBJ**
  - Specification verification in an appropriate abstraction level

Proof Scores in CafeOBJ

- A **proof score** is an **executable high level description of an exhaustive symbolic test** (i.e. a complete proof) in reduction rules (i.e. equations) and reduction commands. That is, a proof score is doing a **proof by reduction**.
- A proof score for a specification (a CafeOBJ module) is a piece of code that constructs **proof modules** and does **reductions** with the equations in the specification and the proof modules.
- **Domain/requirement/design engineers** using CafeOBJ can construct proof scores for specification verification.

CafeOBJ Proof Score 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 protocols
  e.g. NSLPK, Otway-Rees, STS protocols
- Practical sized e-commerce protocol of SET
- UML semantics (class diagram + OCL-assertions)
- Formal Fault Tree Analysis
- Secure workflow models, internal control
- Automatic non-stop software update protocols/systems
- International standard for automotive software
- Protocols for Cloud computing

PTcalc and WFI

- Major high level planing in proof scores includes **lemma**, **case-split**, and **induction**. They are supposed to be found by humans and declared as proof modules.
- Proof modules can be constructed manually with `open-close` constructs, and case-split/induction can be described in a liberal way. It involves, however, **risks of obscuring the rationale of the high level planning**.
- **Proof tree calculus (PTcalc)** and **well-founded induction (WFI)** are incorporated to solve the issue, by realizing a **case-split with exhaustive equations** and an **induction via a well-founded relation**.

PTcalc (Proof Tree Calculus) (1)

- A CafeOBJ's specification (a module) M contains **equations** and defines the set of models $\mathbb{M}od(M)$ each element of that satisfies the equations.
- **PTcalc** (Proof Tree Calculus) is a subsystem of CafeOBJ and helps to prove that a property p holds for any model in $\mathbb{M}od(M)$ (in symbols $M \models p$; M satisfies p). p is expressed as a **Boolean ground term** with fresh constants that correspond to the parameters of the property.

PTcalc (2)

- If the term p is deduced to be equal to `true` with the M's equations (in symbols $M \vdash_{\mathrm{e}} p$), any model of M satisfies p (in symbols $M \models p$) by the soundness of equational deduction (in symbols $M \vdash_{\mathrm{e}} p \Rightarrow M \models p$).
- If the term p is reduced to `true` by using the M's equations as reduction/rewrite rules from left to right (in symbols $M \vdash_{\mathrm{r}} p$), $M \vdash_{\mathrm{e}} p$ holds by the soundness of reduction with respect to equational deduction (in symbols $M \vdash_{\mathrm{r}} p \Rightarrow M \vdash_{\mathrm{e}} p$).
- CafeOBJ's reduction is an implementation of $M \vdash_{\mathrm{r}} p$ and we get $M \vdash_{\mathrm{c}} p \Rightarrow M \vdash_{\mathrm{r}} p$. Because implication ($\Rightarrow$) is transitive the following proof rule is obtained.

  (1) $\quad M \vdash_{\mathrm{c}} p \Rightarrow M \models p$

  (if p reduced to `true` by CafeOBJ at M then M satisfies p)

PTcalc (3)

- Usually $M \vdash_{\mathrm{c}} p$ is difficult to prove directly, and we need to find case splitting equations $e_1, \cdots, e_n$ such that at least one of them holds for any model in $\mathbb{M}od(M)$. That is, the equations cover all the possibilities (i.e. are exhaustive).
- Let M_{+e_i} be the module gotten by adding e_i to M, then each model in $\mathbb{M}od(M)$ is a model of M_{+e_j} for some $j \in \{1, 2, \cdots, n\}$, and we get the following proof rule of **case split with exhaustive equations**.

$$\textbf{(2)} \quad (M_{+e_1} \models p \wedge M_{+e_2} \models p \wedge \cdots \wedge M_{+e_n} \models p) \Rightarrow M \models p$$

PTcalc (4)

- $M_{+e_i} \vdash_{\mathrm{c}} p$ would be still difficult to prove and **(2)** is applied repeatedly. The repeated applications of **(2)** generate **proof trees** successively.
- Each of the generated proof trees has the **root node** $M \models p$ and each of other **nodes** is of the form $M_{+e_{i_1} \cdots +e_{i_m}} \models p$ $(m \in \{1, 2, \cdots\})$ that is generated as a **child node** of $M_{+e_{i_1} \cdots +e_{i_{m-1}}} \models p$ by applying **(2)**.
- A **leaf node** (i.e. a node without child nodes) $M_{+e_{l_1} \cdots +e_{l_k}} \models p$ $(k \in \{0, 1, \cdots\})$ of a proof tree is called **effective** if $M_{+e_{l_1} \cdots +e_{l_k}} \vdash_{\mathrm{c}} p$ holds.
- A proof tree is called **effective** if all of whose leaf nodes are effective. PTcalc proves $M \models p$ by constructing an effective proof tree whose root node is $M \models p$.

```
> root    ⇒      root    ⇒     root      ⇒      root*
                 1*              1*              1*
               > 2               2*              2*
                 3               2-1*            2-1*
                                 2-1-1*          2-1-1*
                                 2-1-2*          2-1-2*
                                 2-1-2-1*        2-1-2-1*
                                 2-1-2-2*        2-1-2-2*
                                 2-2*            2-2*
                               > 3               3*
                                                 3-1*
                                                 3-1-1*
                                                 3-1-1-1*
                                                 3-1-1-1-1*
                                                 3-1-1-1-2*
                                                 3-1-1-2*
                                                 3-1-1-2-1*
                                                 3-1-1-2-2*
                                                 3-1-2*
                                                 3-2*
```

Proof Tree Evolution with PTcalc

An Example of Proof Score about Plus _+_ on Peano Natural Numbers

```
-- PNAT: Peano NATural numbers
mod! PNAT { [Pnat]
op 0 : -> Pnat {constr} .
op s_ : Pnat -> Pnat {constr} .
-- _=_ is built-in
eq (s M:Pnat = s N:Pnat) = (M = N) . }
```

$\texttt{Pnat} \stackrel{\text{def}}{=} \{\texttt{0}, \texttt{s 0}, \texttt{s s 0}, \texttt{s s s 0}, \cdots\}$

```
-- PNAT+ : PNAT with addition _+_
mod! PNAT+ { pr(PNAT)
op _+_ : Pnat Pnat -> Pnat .
vars X Y : Pnat .
eq 0 + Y = Y .
eq s X + Y = s(X + Y) . }
```

A Property of Interest: Associativity of _+_

$\forall x, y, z \in \mathtt{Pnat}\ ((x\ \mathtt{+}\ y)\ \mathtt{+}\ z\ \mathtt{=}\ x\ \mathtt{+}\ (y\ \mathtt{+}\ z))$

```
-- module for defining the associativity of _+_
mod! PNAT+assoc { pr(PNAT+)
pred assoc+ : Pnat Pnat Pnat .
vars X Y Z : Pnat .
eq assoc+(X,Y,Z) = ((X + Y) + Z = X + (Y + Z)) . }
```

for fresh constants "`ops x@ y@ z@ : -> Pnat .`"

"`red in PNAT+assoc : assoc+(x@,y@,x@) .`" implies

$(\mathtt{PNAT+assoc} \models \mathtt{assoc+(x@,y@,z@)})$

Ordinary Induction vs. Well-Founded Induction

- An effective induction scheme varies on problems and an inductive proof score tends to be written in an ad hoc manner.
- Well-founded induction is well known to be powerful enough to subsume a varieties of induction schemes. A method to codify well-founded induction in CafeOBJ proof score has been developed.
- The method makes it possible to codify various induction schemes (including structural induction) in a uniform and transparent style.

Principle of Well-Founded Induction (WFI)

- T : a set.
- p : a predicate on T, i.e. a function from T to the truth values $\{\texttt{true}, \texttt{false}\}$.
- $wf>$: a **well-founded binary relation** on T, i.e. $wf> \subseteq T \times T$ and there is no infinite sequence of $t_i \in T$ with $(t_i, t_{i+1}) \in wf>$ $(i \in \{1, 2, \cdots\})$.

▶ The **principle of WFI** we use is formulated as follows, where $t\ wf> t'$ means $(t, t') \in wf>$.

$$\forall t \in T([\forall t' \in T((t\ wf> t')\ \texttt{implies}\ p(t'))]\ \texttt{implies}\ p(t))\ \texttt{implies}\ \forall t \in T(p(t))$$

I.e., if $p(t)$ whenever $p(t')$ for all $t' \in T$ such that $t\ wf> t'$, then $p(t)$ for all $t \in T$.

Proof of the WFI Principle:

Assume there exits $t \in T$ such that $\texttt{not}(p(t))$, then there should be $t' \in T$ such that $t\ wf{>}\ t'$ and $\texttt{not}(p(t'))$.

Repeating this produces an infinite $wf{>}$-descending sequence. It conflicts with well-foundedness of $wf{>}$.

Proof Module for Defining the Induction Hypothesis of WFI

```
mod PNAT+assoc-wfi-hypo {
pr(PNAT+assoc) -- base module on which the proof applies
-- declaring assoc+'s argument tuple,
-- i.e. a sort Ppp of 3 tuples of Pnat and its constructor t
[Ppp] op t : Pnat Pnat Pnat -> Ppp {constr} .
-- a well-founded binary relation _wf>_ on Ppp
vars X Y Z : Pnat .
pred _wf>_ : Ppp Ppp . eq t(s X,Y,Z) wf> t(X,Y,Z) = true .
-- fresh constants for expressing
-- the goal proposition: assoc+(x@,y@,z@)
ops x@ y@ z@ : -> Pnat .
-- induction hypothesis of WFI
ceq[assoc+ih :nonexec]:
    assoc+(X,Y,Z) = true if t(x@,y@,z@) wf> t(X,Y,Z) .
-- fresh constant for refining term x@
op x$1 : -> Pnat . }
```

Executing Proof by Proof Tree Calculus (PTcalc)

```
select PNAT+assoc-wfi-hypo .
:goal{eq assoc+(x@,y@,z@) = true .}
-- exhaustive equations for refining term x@
:def x@ = :csp{eq x@ = 0 . eq x@ = s x$1 .}
:def ih = :init as assoc+ih-1 [assoc+ih] .
:apply(x@ rd- ih rd-)
```

```
PNAT+assoc-wfi-hypo> :show proof
root*
[x@]   1*
[x@]   2*
[ih]   2-1*
```

$$(\mathrm{PNAT{+}assoc\text{-}wfi\text{-}hypo} \models \mathrm{assoc{+}(x@,y@,z@)}) \Rightarrow (\mathrm{PNAT{+}assoc} \models \mathrm{assoc{+}(x@,y@,z@)})$$

PTcalc+WFI

- PTcalc is more fundamental than WFI and there are many PTcalc proof scores without WFI. However, a **term refinements** (an instance of case-split with exhaustive equations) is necessary to do WFI, and WFI is always PTcalce+WFI.

System Specification in CafeOBJ: ADT or Transition System

- A **static system** is specified as an **Abstract Data Type (ADT)**.
  - ADT is described algebraically with possibly **Conditional Equations**.
- A **dynamic system** is specified as a **Transition System** (i.e., a state transition system).
  - A transition system is described as (i) a special ADT called **OTS (observational transition system)** or (ii) a **Transition Specification**.

A Transition Specification is defined by giving the following three items

- ADT for defining **State Configurations** of the transition system.
- **Transitions** on the state configurations that define the behavior of the transition system. The transitions on the state configurations are defined with possibly conditional **Transition Rules**; a transition rule is a **Special Equation** on state configurations that defines transitions from an instance of the rule's left hand side to the corresponding instance of the rule's right hand side.
- **Initial States** as a subset of the state configurations.

Models of a CafeOBJ Module: Algebras

- For formally verifying specifications, **mathematical models** of a specification (i.e., a CafeOBJ Module) should be defined.
- A mathematical system that consists of a collection of **sets and functions on the sets** is called an **algebra**.
- Each CafeOBJ module denotes a class of algebras each of which interprets the **sorts and operators** declared in the module as **sets and functions on the sets** (i.e., the sorts denote the sets and the operators denote the functions on the sets).

Models of a CafeOBJ Module – Equation, State Transition –

- If **equations** are declared in a module, each model (algebra) A of the module satisfies the equations (i.e., any assignment of A's values to variables makes the left and right hand sides of each equation equal).
- If **a special sort State** and a set TI of **transition rules** on the sort `State` are declared in a module, for each model (algebra) A of the module, TI defines the **transition relation** $TR_{TI} \subseteq A_{\texttt{State}} \times A_{\texttt{State}}$, where $A_{\texttt{State}}$ is the set the sort `State` denotes.

$M \models bt$

"M **satisfies** bt" **or** "bt **is** `true` **for every model of** M"

- CafeOBJ has a builtin module `BOOL`, with a sort `Bool`, that implements the **propositional calculus** as an executable Boolean algebra. A standard CafeOBJ module includes `BOOL` automatically and every model of the module includes the Boolean algebra.
- For a CafeOBJ module M and a ground term (i.e., a term without variables) bt of the sort `Bool`, let $M \models bt$ mean that every model of M **satisfies** bt (i.e., bt is `true` for every model of M). A ground term can contain **fresh constants** each of which denotes, like a variable, any element of the corresponding sort .

Property Specifications and Proof Modules for ADT

- Let $M \models_{\mathrm{prp}} \pi$ mean that a property π holds for every model of a module M.
- If M specifies an ADT, the property π is supposed to be such that a Boolean ground term bt and a module M^{π}, which satisfies the following implication, can be constructed.

  (pm) $M^{\pi} \models bt \Rightarrow M \models_{\mathrm{prp}} \pi$
- Let M_{bt} be the module defining the operators (functions including predicates) needed to express the term bt.

  $M^{\pi} = M + M_{bt}$

  M^{π}, M_{bt} : **proof modules**

  M_{bt} : **property specification**

Properties for Transition Specifications

- If M is a transition specification and specifies a transition system, π is supposed to be an **invariant property** or a **leads-to property**.
- An **invariant property** is a safety property and is specified with a state predicate p that holds for all reachable states.
- A **leads-to property** is a liveness property and is specified with two state predicates p and q.
  A **(p leads-to q) property** holds for a transition system if the system's entry to the state where p holds implies that the system will surely get into the state where q holds no matter what transition sequence is taken.

Property Specifications and Proof Modules for TSs

- For a transition specification (module) M and its property π, modules M_i^π and Boolean ground terms bt_i $(1 \leq i \leq m)$ that satisfy the following implication can be constructed by making use of **builtin search predicates**.

  (PM) $(M_1^\pi \models bt_1 \wedge M_2^\pi \models bt_2 \wedge \cdots \wedge M_m^\pi \models bt_m) \Rightarrow M \models_{\mathrm{prp}} \pi$
- Let M_{bt_i} be the property specification defining the operators needed to express the term bt_i $(1 \leq i \leq m)$, then $M_i^\pi = M + M_{bt_i}$.
  - Predicates p, q needed to express invariant or leads-to properties are also specified in M_{bt_i} $(1 \leq i \leq m)$.

Model Based Deduction Rule:

(PR1) $M \vdash_{\mathrm{c}} bt \Rightarrow M \models bt$

- CafeOBJ system has a reduction command "`reduce in` M : bt ." for getting a reduced form of bt in the module M by using M's equations as reduction rules from left to right.
- Let $M \vdash_{\mathrm{c}} bt$ means that "`reduce in` M : bt ." returns `true`, then the following primary proof rule says that bt is `true` for every model of M if bt reduces to `true` in M.

(PR1) $\boxed{M \vdash_{\mathrm{c}} bt \Rightarrow M \models bt}$

Model Based Deduction Rule: (PR2)

- Let M be a module with a signature Σ and a set of equations E ($M = (\Sigma, E)$ in symbols). Equations $e_1, \cdots, e_n$ $(1 \leq n)$ are defined to be **exhaustive** for M iff for any model A of M there exists some i $(1 \leq i \leq n)$ such that e_i holds in A.
- Let $e_1, \cdots, e_n$ $(1 \leq n)$ be exhaustive equations and $M_{+e_i} = (\Sigma \cup Y_i, E \cup \{e_i\})$ $(1 \leq i \leq n$, Y_i is the set of fresh constants in $e_i)$, then the following proof rule holds on which every case-split in CafeOBJ can be based.

(PR2) $\boxed{(M_{+e_1} \models bt \wedge M_{+e_2} \models bt \wedge \cdots \wedge M_{+e_n} \models bt) \Rightarrow M \models bt}$

Constractors and Constrained Sorts

- A **constractor** of a sort is an operator that constructs elements of the sort. A sort is called **constrained** if there exists a constructor of the sort. Let $bt(y@_1, \cdots, y@_n)$ be a ground term of the sort `Bool` containing fresh constants $y@_i$ of constrained sorts s_i $(1 \leq i \leq n)$.

Model Based Deduction Rule: (WFI)

- Let $Y@$ be the set $\{y@_1, \cdots, y@_n\}$, $\overline{y@}$ be the n-tuple $y@_1, \cdots, y@_n$, y_i be variables of sorts $s_i (1 \leq i \leq n)$, and $\overline{y}$ be the n-tuple $y_1, \cdots, y_n$. Let $\_wf>\_$ be a well-founded binary relation on the set of n-tuples $ct_1, \cdots, ct_n$ of ground constructor terms of sorts $s_1, \cdots, s_n$, and $M_{wf>}$ be the module adding the definition of $\_wf>\_$ to M. The following **well-founded induction (WFI)** rule can be used for deducing $M \models bt(y@_1, \cdots, y@_n)$.

(WFI)

$$(M_{wf>} \cup Y@ \cup \{\texttt{cq}\ bt(\overline{y}) = \texttt{true if}\ \overline{y@}\ wf>\ \overline{y}\ .\}) \models bt(\overline{y@}) \Rightarrow M \models bt(\overline{y@})$$

PTcalc (Proof Tree Calculus): (PR1,PR2)*

- PTcalc is a subsystem of CafeOBJ system to support proving $M \models bt$ with the proof rules **(PR1)** and **(PR2)**.
- $M \vdash_{\mathrm{c}} bt$ does not always holds, and $M \models bt$ is usually difficult to duduce directly using the proof rule **(PR1)**. Exhaustive equations $e_1, \cdots, e_n$ need to be found and the proof rule **(PR2)** should be used.
- $M_{+e_i} \models bt$ would be still difficult to prove with the proof rule **(PR1)**, and **(PR2)** should be applied repeatedly. The repeated applications of **(PR2)** generate **proof trees** successively.

PTcalc (Proof Tree Calculus): Effective Proof Tree

- Each of the generated proof trees has the **root node** $M \models bt$ and each of other **nodes** is of the form $M_{+e_{i_1} \cdots +e_{i_m}} \models bt$ $(1 \leq m)$ that is generated as a **child node** of $M_{+e_{i_1} \cdots +e_{i_{m-1}}} \models bt$ by applying **(PR2)**.
- A **leaf node** (i.e., a node without child nodes) $M_{+e_{l_1} \cdots +e_{l_k}} \models bt$ $(0 \leq k)$ of a proof tree is called **effective** if $M_{+e_{l_1} \cdots +e_{l_k}} \vdash_{\mathrm{c}} bt$ holds.
- A **proof tree** is called **effective** if all of whose leaf nodes are effective. PTcalc proves $M \models bt$ by constructing an effective proof tree whose root node is $M \models bt$.

A Generic Procedure for Constructing PSs: (1)(2)(a)(i)(ii)(b)

A collection of pieces of CafeOBJ code for deducing that a property π holds for every model of a module M ($M \models_{\mathrm{prp}} \pi$ in symbols) is called a proof score. A generic procedure to construct a proof score for deducing $M \models_{\mathrm{prp}} \pi$ is as follows.

(1) Construct proof modules M_i^π and Boolean ground terms bt_i $(1 \leq i \leq m)$ satisfying the following implication by using the rule **(pm)** or **(PM)**.

$$\boxed{(M_1^\pi \models bt_1 \wedge M_2^\pi \models bt_2 \wedge \cdots \wedge M_m^\pi \models bt_m) \Rightarrow M \models_{\mathrm{prp}} \pi}$$

(2) (a) (i)
(ii)
(b)

A Generic Procedure for Constructing PSs: (1)(2)(a)(i)(ii)(b)

(1)

(2) For each $M_i^{\pi} \models bt_i \, (1 \leq i \leq m)$ do as follows.

(a) Construct a root module M_i^{rt} as follows depending on whether the rule **(WFI)** is used or not.

(i) If bt_i is expressed as $bt_i(\overline{y@})$ with fresh constants $\overline{y@}$ of constrained sorts and the $bt_i(\overline{y@})$'s proof can be achieved with WFI, then define a well-founded relation $\_wf>\_$ appropriately and construct the module M_i^{rt} as follows with variables $\overline{y}$ of the corresponding constrained sorts.

$$M_i^{\mathrm{rt}} = (M_i^{\pi})_{wf>} \cup Y@ \cup \{\mathtt{cq}\ bt_i(\overline{y}) = \mathtt{true\ if}\ \overline{y@}\ wf> \overline{y}\ .\}$$

(ii) If the condition of (i) is not satisfied, then construct the module M_i^{rt} as follows.

$$M_i^{\mathrm{rt}} = M_i^{\pi}$$

(b)

A Generic Procedure for Constructing PSs: (1)(2)(a)(i)(ii)(b)

(1)

(2) (a) (i)

(ii)

(b) Construct a CafeOBJ code with PTcalc commands that constructs an effective proof tree with the root node $M_i^{\mathrm{rt}} \models bt_i$. If a lemma (a property) λ is found necessary to be proved with respect to a module M^λ (i.e., $M^\lambda \models_{\mathrm{prp}} \lambda$ needs to be proved), then set $\pi = \lambda$, $M = M^\lambda$ and construct a proof score for $M \models_{\mathrm{prp}} \pi$ using the procedure (1)(2)(a)(i)(ii)(b). Note that constructions of proof scores for lemmas may be recursively invoked.

The procedure (1)(2)(a)(i)(ii)(b) does not always terminate. If terminates, the CafeOBJ codes constructed, including the codes for proving necessary lemmas, constitute a proof score for deducing $M \models_{\mathrm{prp}} \pi$.

Characteristics of PTcalc+WFI Proof Scores

Comparing with the most notable theorem proving systems like Coq, Isabelle/HOL, PVS, the spec verification with PTcalc+WFI proof scores in CafeOBJ has the following characteristics.

(i) Systems/services are modeled with **Algebraic Abstract Data Types (ADT)** and/or **Transition Specifications** and an appropriate higher abstraction level can be settled for each specification.

(ii) The only two proof rules **(PR1)**, **(PR2)** are simple and transparent, formalized at the level of **specification satisfaction** $SP \models p$ (i.e. any model of SP satisfies p), and supporting **model based proofs**.

Soundness Conditions for PTcalc+WFI Proof Scores

The conditions for PTclac+WFI proof scores to be correct (sound) are intensively localized into the following three points:

(i) **Exhaustiveness of the equations** declared in a `:csp{...}` command;

(ii) **Validity of the equation** declared inside `(...)` of a `:init` command "`:init ... (...) ...`";

(iii) **Well-foundedness of the binary relation** `_wf>_` used to declare an induction hypothesis.

The well-foundedness of a binary relation `_wf>_` on argument n tuples of a goal predicate can be established, in the majority of cases, via **the proper sub-term relaions on constructor terms**. This contributes to applicability and flexibility of PTcalc+WFI.

Proof Automation and Proof Planning in PTcalc+WFI

- Fully automatic verifications often fail to convey important properties/structures of the systems/problems.
- **Balanced optimal** of **proof automation** and **proof planning** is important; Computers for the tedious formal calculations, and humans for the high level planning. Proof scores intend to meet these requirements.
- In **PTcalc+WFI**, proof automation is achieved by construction of proof trees with effectiveness checks via **fully automated reductions**, and Proof planning is codified into **proof modules** and **reduction commands**.
- This **two layered architecture** is simple, but powerful enough to achieve a nice balance of proof automation and proof planning.

Future Perspective

- There should be many chances of developing significant practical specifications and proof scores in PTcalc+WFI.
- The current CafeOBJ system provides a nice platform for conceiving, experimenting, and/or formalizing new specification verification methods on top of proof scores in PTcalc+WFI.

You can find almost all necessary information on CafeOBJ at:

https://cafeobj.org/

Slides for this talk is at:

https://cafeobj.org/~futatsugi/talk/lac2025/

A recently published historical and comprehensive survey:

Proof Scores: A Survey
by A Riesco, K Ogata, M Nakamura, D Gaina, D D Tran, K Futatsugi,
ACM Computing Surveys, 57-10, Article 251, pp.1-37, May 2025
https://doi.org/10.1145/3729166