# TPP 2019 Mark in CafeOBJ - as an Example of Specification Verification -

FUTATSUGI,Kokichi
二木 厚吉

JAIST(Japan Advanced Institute of Sciecne and Technology)
AIST (National Institute of Advanced Industrial Science and Technology)

## What is Specification Verification?

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

- **"All models are wrong, some are useful."**
- **Specification Verification** is a good (may be the strongest) way for getting high-quality specifications (useful descriptions of models).
- **Specification Verification** is still one of the most challenging topics in software/system engineering.

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

- CafeOBJ is an executable algebraic specification language system which can be used as a specification verification system.

**Proof Score Approach in CafeOBJ**

- Domain/requirement/design engineers are expected to construct proof scores (descriptions of proofs in CafeOBJ) together with formal specifications.
- A proof score is an executable high level description
  - of an exhaustive symbolic test (i.e. a complete proof)
  - in equations (reduction rules) and reduction commands.

  **Proof by computation/reduction/rewriting**

## Theorem Proving vs Specification Verification

- The primary goal of **Theorem Proving** is to show that a theorem can be deduced from axioms with deduction rules. Usually the axioms are already established ones.
- The primary goal of **Specification Verification** is to show that a specification is complete enough to imply/infer desirable properties. The specification (a set of axioms) are supposed to be improved/refined through specification verification.

## Specification Verification and Proof by Reduction

- The final goal of **Specification Verification** is to get a high-quality specification (a set of axioms) which is reliable enough to build important systems/services based on it.
- The main technical issue is systematic construction of a reliable specification (a set of axioms).
- **Proof by reduction/rewriting** is a transparent deduction/inference method, and contributes to ease the inherent difficulty of **Specification Verification**.

## Answer 1

```
1. Please formally define a notion of Turing
machines that manipulate cyclic tapes. A cyclic tape is a
fixed-length tape whose both ends are connected to each
other. You can base on any paper definition of Turing
machines.
-->
--> Answer to 1.
--> The definition (specification) of Turing Machine with
--> Cyclic Tape (TMCT) is given in the file 'tmct.cafe'.
```

## TMCT Transition Rules

```
ops ct nt : -> State . ops cs ns : -> Symbol .
vars SS1 SS2 : SeqSym . var SY : Symbol .
tr[l-1]: SS1 SY | ct cs | SS2 =>
            SS1 | nt SY | ns SS2 .
tr[l-2]: nilsym | ct cs | SS SY =>
           ns SS | nt SY | nilsym .
tr[nm]:  nilsym | ct cs | nilsym =>
         nilsym | nt ns | nilsym .
tr[r-1]:    SS1 | ct cs | SY SS2  =>
         SS1 ns | nt SY | SS2 .
tr[r-2]:  SY SS | ct cs | nilsym =>
         nilsym | nt SY | SS ns .
```

## Answer 2

```
2. Please definetheir one-step computation.
-->
--> Answer to 2.
--> Transition rules defined in the modules
--> TMCT-l-1 (tr[l-1]), TMCT-l-2 (tr[l-2]), TMCT-nm (tr[nm]),
--> TMCT-r-1 (tr[r-1]), TMCT-r-2 (tr[r-2])
--> define one-step computation of TMCT.
--> A concrete TMCT is specified as a finite set of
--> transition rules each element of which is obtained from
--> one of tr[l-1], tr[l-2], tr[nm], tr[r-1]), or tr[r-2]
--> by instantiating ct, cs, nt, and ns.
```

## Answer 3

```
3. Please define the predicate that a machine halts on
   a given input tape.
-->
--> Answer to 3.
--> Let TMCT-1 be the given TMCT, and let cf be the
--> configuration of TMCT-1 with the given input tape.
--> Then TMCT-1 starting from cf halts if and only if
--> the following reduction halts.
-->       reduce in TMCT-1 : cf =(*,*)=>* CF:Cnfg .
--> Here the reduction successively prints out all the
--> reachable configurations from cf.
```

## Answer 4

```
4. Please define a machine that, given an
input cyclic tape of unknown size and content, clears the
tape and halts. You may assume a minimum length on input
tapes. This puzzle was given to me by Vincent van Oostrom.
-->
--> Answer to 4.
--> TMCT CLR-l clears input symbols towards left and halts,
--> and CLR-r clears input symbols towards right and halts.
--> CLR-l and CLR-r are specified in the file 'clr-lr.cafe'
```

## Answer 5

```
5. Please demonstrate how your machine above works on a
concrete tape example.
-->
--> Answer to 5.
--> Examples of transitions (computations) of CLR-l, CLR-r,
--> and (CLR-l + CLR-r) are given in
--> the file clr-lr-tst.cafe.
```

## Answer 6 (1)

```
6. (Optional) Please prove that your machine halts and
resulting tape is cleared for any input cyclic tape.
-->
--> Answer to 6.
--> Proof scores in the files tmct-szeqsz-ps.cafe,
--> cnfg-is-istsis-ps.cafe, clr-l-tsis-lt-ts-ps.cafe,
--> clr-r-ists-lt-ts-ps.cafe, and clr-lr-ts-imp-halt-ps
--> prove that each of CLR-l and CLR-r clears input tape
--> and halts with blank tape.
```

## Target systems

- Target systems are supposed to be modeled as (possibly infinite state) transition systems.
- A transition system is specified with the following two entities.
  1. **Abstract data types for defining state configurations** of the transition system. The abstract data types are specified in equations which can also be conditional.
  2. **Transitions that define behavior of the transition system**. The transitions are specified in transition rules which can also be conditional.

## Target properties

The desirable properties of a transition system that are supposed to be verified are **invariant properties or leads-to properties**.

- An **invariant property** is defined as a state predicate that holds for all reachable states from initial states.
- A **leads-to property** is defined as a pair of state predicates ($p$,$q$), and defined to hold for a 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. **($p$ leads-to $q$)** denotes that ($p$,$q$) is a leads-to property.

## Basic leads-to properties

A **Basic leads-to property** is defined as a pair of state predicates $(p,q)$, and defined by the following two items. **($p$ b-leads-to $q$)** denotes that $(p,q)$ is a basic leads-to property.

- If $s$ is a reachable state with $(p(s)$ `and` $($`not` $q(s)))$, then (1) $s$ has at least one next state, and (2) $(p(s')$ `or` $q(s'))$ holds for any next state $s'$ of $s$.
- There exits no infinite transition sequence $sq$ such that (1) it starts with a reachable state, and (2) $(p(s)$ `and` $($`not` $q(s)))$ holds for any element state $s$ of $sq$.

## Verification Conditions for Basic leads-to Properties

Let $p$,$q$ be state predicates of a transition system $T$, let `State` be set of states of $T$, let $\texttt{TR}_T \subseteq \texttt{State} \times \texttt{State}$ be the set of one-step transitions of $T$, and let `Nat` be the set of natural numbers.

If an invariant *inv* of $T$ and an operator $m$ of rank '`State -> Nat`' are defined such that the following two conditions hold, then ($p$ b-leads-to $q$) holds.

(1) $(\forall (s,s') \in \texttt{TR}_T$
$((inv(s)$ and $p(s)$ and $(\texttt{not}\ q(s)))$ $\texttt{implies}$
$((p(s')$ $\texttt{or}$ $q(s'))$ and $(m(s) > m(s')))))$

(2) $(\forall s \in \texttt{State}$
$((inv(s)$ and $p(s)$ and $(\texttt{not}\ q(s)))$ $\texttt{implies}$
$(\exists s' \in \texttt{State}((s,s') \in \texttt{TR}_T))))$

(1) is called **inductive leads-to condition**, and condition (2) is called **continuous leads-to condition**.

## Verification Conditions for leads-to Properties

Let $p$,$q$,$r$ be state predicates of a transition system $T$, let ($p$ `-b>`$_T$ $q$) denote that ($p$ b-leads-to $q$) holds for $T$, and let ($p$ `->`$_T$ $q$) denote that ($p$ leads-to $q$) holds for $T$.

The following 4 items give verification consitions of ($p$ `->`$_T$ $q$) based on ($p$ `-b>`$_T$ $q$).

(1) ($p$ `-b>`$_T$ $q$) `implies` ($p$ `->`$_T$ $q$)
(2) (($p$ `->`$_T$ $q$) `and` ($q$ `->`$_T$ $r$)) `implies` ($p$ `->`$_T$ $r$)
(3) (($p$ `->`$_T$ $r$) `and` ($q$ `->`$_T$ $r$)) `implies` (($p$ or $q$) `->`$_T$ $r$)
(4) (($p$ `->`$_T$ $r$) `or` ($q$ `->`$_T$ $r$)) `implies` (($p$ and $q$) `->`$_T$ $r$)

## Answer 6 (2)

```
--> The file tmct-szeqsz-ps.cafe proves that
--> for any configuration cf of any TMCT,
-->      (size(cf) = size(cf’))
--> holds for any configuration cf’ that is reachable
--> from cf.
require tmct-szeqsz-ps
```

## Answer 6 (3)

```
--> The file cnfg-is-istsis-ps.cafe proves that
-->      (is..(_:Cnfg) leads-to tsis(_:Cnfg))
--> and
-->      (is..(_:Cnfg) leads-to ists(_:Cnfg))
--> hold for any TMCT including CRL-l and ClR-r.
require cnfg-is-istsis-ps
```

## Answer 6 (4)

```
--> The file clr-l-tsis-lt-ts-ps.cafe proves that
-->    (tsis(_:Cnfg) leads-to ts..(_:Cnfg))
--> holds at the CRL-l.
require clr-l-tsis-lt-ts-ps
-->
--> The file clr-r-ists-lt-ts-ps.cafe proves that
-->    (ists(_:Cnfg) leads-to ts..(_:Cnfg))
--> holds at the CRL-r.
require clr-r-ists-lt-ts-ps
```

## Answer 6 (5)

```
--> The file clr-lr-ts-imp-halt-ps.cafe proves that
-->  (ts..(_:Cnfg) leads-to (ts..(_:Cnfg) and halt(_:Cnfg)))
--> holds for CLR-l and CLR-r.
require clr-lr-ts-imp-halt-ps.cafe
```

## Answer 6 (6)

```
--> Hence it is proved that
-->  (is..(_:Cnfg) leads-to (ts..(_:Cnfg) and halt(_:Cnfg)))
--> holds for CLR-l and CLR-r.
--> It implies that if CLR-l or CLR-r starts with a
--> configuration cf such that (is..(cf) = true),
--> i.e. cf with a tape with only input symbols,
--> then it surely halt with a configuration cf'
--> such that (ts..(cf') = true),
--> i.e. cf' with a tape with only tape symbols,
--> no matter what transition sequence it takes.
```

## 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 protocol
  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

## Coordinating Deduction and Search

- Built-in search predicates provide effective ways to do proof (1) by search for cases of small size, and (2) by deduction for cases of large or infinite size .
- The transparent proof structure has a potential to make "lemma finding" easier by analyzing the problem domain but not detailed logical inference rules.
- "PTG (Proof Tree Generation) + RC/RPC (Refinement of Cases through Refinement of Proof Constants)" provide an effective way to do deduction (proof tree generation) with systematic search induced by refinement of state configurations (through refinement of proof constants).

## Well-Founded Induction

Let $T$ be a set of elements with a **well-founded** binary relation $<_{wf}$. That is, there is no infinite sequence of elements $t_1, t_2, t_3, \cdots$ such that $t_{i+1} <_{wf} t_i$ $(i \in \{1, 2, 3 \cdots\})$.
The principle of well-founded induction is stated as follows.

**[Principle of Well-Founded Induction (Noetherian Induction)]**
Let $p$ be a predicate on $T$ then

$$(\forall t \in Srt((\forall t' <_{wf} t(p(t'))) \,\texttt{i}mplies\, p(t)))$$
$$\texttt{i}mplies\, (\forall t \in Srt(p(t))).$$

That is, if $p(t)$ whenever $p(t')$ for all $t' \in T$ such that $t' <_{wf} t$, then $p(t)$ for all $t \in T$.

## Well-Founded Induction in CafeOBJ

- The **well-founded induction** in CafeOBJ combines the term refinement of and the induction on `c1,c2,..,cn`, and has a potential to support a variety of induction schemes.
- **Theorem of Constants**, **Casesplit with Equations**, **Term Refinement with Equations**, **Specification Calculus** in CafeOBJ justify the well-founded induction naturally.

ライブラリ｜情報学コア・テキスト13
モデルの記述と検証のための
プログラミング入門
―CafeOBJによる仕様検証―
二木 厚吉 〔著〕
サイエンス社

Introduction to Specification Verification with CafeOBJ