A Brief Overview of CafeOBJ/ProofScore and Formal Methods

FUTATSUGI, Kokichi
二木 厚吉

JAIST-RCSV
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 (models/specifications 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
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 not necessary well accepted to software/systems engineers
  - especially in upstream (pre-coding) phase
Our approach

♦ Reasonable blend of user and machine capabilities, intuition and rigor, high-level planning and tedious formal calculation
  • fully automated proofs 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
Proof Score Approach

- Domain/requirement/design engineers are expected to construct proof scores together with formal specifications.

- Proof scores are instructions such that when executed (or "played") and everything evaluates as expected, then the desired property is convinced to hold.
  - Proof by construction/development
  - Proof by reduction/computation/rewriting
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 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
A little bit of CafeOBJ history

- KF thought of the basic ideas of CafeOBJ after he participated OBJ project at SRI in 1983-1984, and several design and implementation attempts were done during 1985-1995
- The CafeOBJ development project is fully supported by IPA/MITI of Japanese Government from 1996.4 to 1998.3
  - Six Japanese Companies, Five Japanese Universities, Three Foreign Research Group participate CAFÉ project
  - A book entitled “CafeOBJ Report” was published in 1998 which defines the syntax and semantics of the CafeOBJ language
- Sufficiently reliable and usable CafeOBJ system was available at around the beginning of 1999.
- Several groups including KF’s group at JAIST are using CafeOBJ for developing formal methods for various application areas and/or for education of FM
An Overview of KAKEN-KIBAN(S) Project

2011.4 - 2016.3

http://www.jsps.go.jp/j-grantsinaid/12_kiban/ichiran_23/e-data/e02_futatsugi.pdf
An Important Key Technology in Software Engineering

Software has became an important infrastructure of our society, and reliability and security are most important attributes.

- middleware software/protocol
  (electric commerce, cloud computing/services)
- International standards for important software systems
  (standards for on-board computers, etc.)

Verification of Problem Specifications is a Key Technology

Verification Language/System of Problem Specs
Problem Verification with Proof Scores

Technical Issues

1. Development of specs in appropriate abstraction level
2. Verification methods combining inference and search

CafeOBJ Proof Scores

Innovate Spec Verification Methods and Practical Problem Spec Verification System
Concurrent development of cases and methods

Requirements from cases

Practical cases
- on-board OS standard
- middleware protocol

Problem spec Verification System (CafeOBJ System)

Verification with PS

Innovative problem spec verification methods
- appropriate abstraction
- inference x search
Generate & Check Method for Verifying Transition Systems in CafeOBJ (an overview)

Kokichi FUTATSUGI

Research Center for Software Verification
Japan Advanced Institute of Science and Technology

30 October 2014
JAIST Logic and Verification Joint Workshop, Kaga, Japan
A Interactive Theorem Proving Method for Verifying Transition Systems

- The state space is defined as a quotient set of terms of a top most sort State.
- The transitions are defined with conditional rewrite rules over the quotient set.
Properties to be Proved

- Invariant properties (state predicates valid for all reachable states)

- \((p \ leads-to \ q)\) property for two state predicates \(p\) and \(q\). \((p \ leads-to \ q)\) means that from any reachable state \(s\) with \((p(s) = true)\) the system will get into a state \(t\) with \((q(t) = true)\) no matter what transition sequence is taken.
Constructing Proof Score for Generate & Check

Verification is achieved by developing proof scores in CafeOBJ. Sufficient verification conditions are formalized for verifying invariants and $(p \rightarrow q)$ properties.

1. Generate a finite set of state patterns that covers all possible infinite states,
2. Check validity of the verification condition for all the covering state patterns by reductions.
Let $TR = \{tr_1, \cdots, tr_m\}$ be a set of transition rules, let $\rightarrow_{TR} \overset{\text{def}}{=} \bigcup_{i=1}^{m} \rightarrow_{tr_i}$, and let $In \subseteq (T_{\Sigma})_{\text{State}}/(=E)_{\text{State}}$.

Then $(\Sigma, E, TR)$ defines a transition system 

$$(((T_{\Sigma})_{\text{State}}/(=E)_{\text{State}}, \rightarrow_{TR}, In))$$

Note that $(T_{\Sigma})_{\text{State}}/(=E)_{\text{State}}$ is better to be understood as $T_{\Sigma}/=E$, for usually the sort State can only be understood together with other related sorts like Bool, Nat, Queue, etc.

A specification like $(\Sigma, E, TR)$ is called a transition specification.
Global view of QLOCK as an Observational Transition System
-- wt: want transition
mod! WT {pr(STATE)
trans[wt]: (Q:Qu $ ((lb[A:Aid]: rs) AS:Aobs))
    => ((Q & A) $ ((lb[A]: ws) AS)). }

-- ty: try transition
mod! TY {pr(STATE)
trans[ty]: ((A:Aid & Q:Qu) $ ((lb[A]: ws) AS:Aobs))
    => ((A & Q) $ ((lb[A]: cs) AS)). }

-- ex: exit transition
mod! EX {pr(STATE)
ctrans[ex]: ((A1:Aid & Q:Qu) $ ((lb[A2:Aid]: cs) AS:Aobs))
    => (Q $ ((lb[A2]: rs) AS))
    if (A1 = A2). }

-- system specification of QLOCK
mod! QLOCKsys{pr(WT + TY + EX)}
$TS = (St, Tr, In), (\forall s \in St)(init(s) \text{ iff } (s \in In))$
p_1, p_2, \cdots, p_n (n \in \{1, 2, \cdots\}) be state predicates of $TS$

[Invariant Lemma] The following three conditions are sufficient for $p_t$ to be an invariant. Here, $inv(s) \overset{\text{def}}{=} (p_1(s) \text{ and } p_2(s) \text{ and } \cdots \text{ and } p_n(s))$ for $s \in St$.

(1) $(\forall s \in St)(inv(s) \text{ implies } p_t(s))$
(2) $(\forall s \in St)(init(s) \text{ implies } inv(s))$
(3) $(\forall (s, s') \in Tr)(inv(s) \text{ implies } inv(s'))$

A predicate that satisfies the conditions (2) and (3) like $inv$ is called an **inductive invariant**. To find $p_1, p_2, \cdots, p_n$ is a most difficult part of the invariant verification.
It is worthwhile to note that there are following two contrasting approaches for formalizing $p_1, p_2, \cdots, p_n$ for a transition system and its property $p_t$.

- **Make $p_1, p_2, \cdots, p_n$ as minimal as possible to imply the target property $p_t$;**
  - usually done by lemma finding in interactive theorem proving,
  - it is difficult to find lemmas without some comprehensive understanding of the system.

- **Make $p_1, p_2, \cdots, p_n$ as comprehensive as possible to characterize the system;**
  - usually done by specifying elemental properties of the system as much as possible in formal specification development,
  - it is difficult to identify the elemental properties without focusing on the property to be proved (i.e. $p_t$).
A term \( t' \in T_\Sigma(Y) \) is defined to be an **instance** of a term \( t \in T_\Sigma(X) \) iff there exits a substitution \( \theta \in T_\Sigma(Y)^X \) such that \( t' = t \theta \). A finite set of terms \( C \subseteq T_\Sigma(X) \) is defined to **subsume** a (may be infinite) set of ground terms \( G \subseteq T_\Sigma \) iff for any \( t' \in G \) there exits \( t \in C \) such that \( t' \) is an instance of \( t \).

**Subsume Lemma** Let a finite set of state terms \( C \subseteq T_\Sigma(X)_{\text{State}} \) subsume the set of all ground state terms \( (T_\Sigma)_{\text{State}} \), and let \( p \) be a state predicate.

\[
(\forall s \in C)(p(s) \rightarrow_E^* \text{true}) \implies (\forall t \in (T_\Sigma)_{\text{State}})(p(t) \rightarrow_E^* \text{true})
\]
\((t_1 \rightarrow_E^* t_2)\) means that the term \(t_1\) is reduced to the term \(t_2\) by the CafeOBJ's reduction engine.

\((t_1 \rightarrow_E^* t_2)\) implies \((t_1 \rightarrow_E^* t_2)\) but not necessary \((t_1 \rightarrow_E^* t_2)\) implies \((t_1 \rightarrow_E^* t_2)\).

**Generate&Check-S** Let \(\left( (T_\Sigma)_{\text{State}}/(=E)_{\text{State}}, \rightarrow TR, ln \right)\) be a transition system. The following generate & check are sufficient for verifying

\[(\forall t \in (T_\Sigma)_{\text{State}})(p_{st}(t) =_E true)\]

for a state predicate \(p_{st}\).

- **Generate** a finite set of state terms \(C \subseteq T_\Sigma(X)_{\text{State}}\) that subsumes \((T_\Sigma)_{\text{State}}\).

- **Check** \((p_{st}(s) \rightarrow_E^* true)\) for each \(s \in C\).
For translating ($\forall tr \in Tr$) to ($\forall st \in St$) for ($\Sigma, E, TR$).

$$\text{pred } _{=}(*,1)=>+\_\text{if}\_\text{suchThat}\_{\_} : \text{State State Bool Bool Info}.$$  

For a state term $s \in T_{\Sigma}(Y)_{\text{State}}$, the reduction of a Boolean term:

$$s =(*,1)=>+ \; SS:\text{State if} \; CC:\text{Bool}$$

$$\text{suchThat} \; p(s,SS,CC) \; \{i(s,SS,CC)\}$$

with $\rightarrow^*_{E \cup \rightarrow TR}$ is defined to behave as follows.
1. Search for every pair \((tr_j, \theta)\) of a transition rule
   \(tr_j = (\forall X)(l_j \rightarrow r_j \text{ if } c_j)\) in \(TR\) and a substitution
   \(\theta \in T_\Sigma(Y)^X\) such that \(s = l_j \theta\).

2. For each found \((tr_j, \theta)\), let \((SS = r_j \theta)\) and \((CC = c_j \theta)\) and
   print out \(i(l_j \theta, r_j \theta, c_j \theta)\) and \(tr_j\) if \((p(l_j \theta, r_j \theta, c_j \theta) \xrightarrow{E}^*\) true\) holds.

3. Returns \text{true} if some print out exits, and returns \text{false}
   otherwise.
Let \( q \) be a predicate “\( \text{pred } q : \text{State } \text{State} \)” for stating some relation of the current state and the next state.

Let the predicates \( \_\_\text{then}_\_ \) and \( \text{valid-q} \) be defined as follows in CafeOBJ using the built-in search predicate

\[
\text{pred } \_\_\text{then}_\_ : \text{Bool } \text{Bool} .
\]
\[
\text{eq } (\text{true then } B:\text{Bool}) = B .
\]
\[
\text{eq } (\text{false then } B:\text{Bool}) = \text{true} .
\]
\[
\text{pred } \text{valid-q} : \text{State } \text{State } \text{Bool} .
\]
\[
\text{eq } \text{valid-q}(S:\text{State},SS:\text{State},CC:\text{Bool}) =
\]
\[
\text{not}(S =(*,1)\Rightarrow SS \text{ if } CC
\]
\[
\text{suchThat } \text{not}((CC \text{ then } q(s,SS)) == \text{true})
\]
\[
\{i(S,SS,CC)} .
\]
[Definition of Cover] Let $C \subseteq T_\Sigma(Y)$ and $C' \subseteq T_\Sigma(X)$ be finite sets. $C$ is defined to cover $C'$ iff for any ground instance $t'_g \in T_\Sigma$ of any $t' \in C'$, there exits $t \in C$ such that $t'_g$ is an instance of $t$ and $t$ is an instance of $t'$.

[Cover Lemma 1] Let $C' \subseteq T_\Sigma(X)_{\text{State}}$ be the set of all the left hand sides of the transition rules in $TR$, and let $C \subseteq T_\Sigma(Y)$ cover $C'$, then the following holds.

\[(\forall t \in C)(\text{valid-q}(t, SS:\text{State}, CC:\text{Bool}) \rightarrow^*_E \cup \rightarrow_{TR} \text{true})\]

implies
\[(\forall (s, s') \in ((T_\Sigma \times T_\Sigma) \cap \rightarrow_{TR}))(q(s, s') \rightarrow^*_E \text{true}))\]
Overview and Preliminaries
Specification of QLOCK in CafeOBJ
Generate & Check Method
Conclusion

Generate & Check for $\forall st \in St$
Built-in Search Predicate of CafeOBJ
Generate & Check for $\forall tr \in Tr$
Generate & Check for Verification of Invariant Properties
Verification of ($p \text{ leads-to } q$) Properties
Generate & Check for Verification of ($p \text{ leads-to } q$) Properties

Covering

\[ S: State \]

\[ \text{concrete states} \]

\[ \text{instances} \]

...
The [Cover Lemma 1] and the [Reduction Lemma] imply the validity of the following generate & check.

**Generate&Check-T1** Let \(((T_\Sigma)_{\text{State}}/ (\equiv E)_{\text{State}}, \rightarrow TR, In)\) be a transition system defined by a transition specification \((\Sigma, E, TR)\), and let \(C' \subseteq T_\Sigma(X)\) be the set of all the left hand sides of the transition rules in \(TR\). The following generate & check are sufficient for verifying

\[
(\forall (s, s') \in ((T_\Sigma \times T_\Sigma) \cap \rightarrow TR))(q_{tr}(s, s') = E \text{ true})
\]

for a predicate “\(\text{pred q}_{tr} : \text{State State}\)”.

**Generate** a finite set of state terms \(C \subseteq T_\Sigma(Y)_{\text{State}}\) that covers \(C'\).

**Check** \((\text{valid-q}_{tr}(t, SS:\text{State}, CC:\text{Bool}) \longrightarrow^*_E \cup \rightarrow TR \text{ true})\) for each \(t \in C\).
The condition (1) and (2) of [Invariant Lemma] can be verified by using Generate&Check-S with $p_{st}(s)$ defined as follows.

\[
\begin{align*}
(1) & \quad p_{st}(s) = (inv(s) \text{ implies } p_t(s)) \\
(2) & \quad p_{st}(s) = (init(s) \text{ implies } inv(s))
\end{align*}
\]

Note that, if $inv \overset{\text{def}}{=} (p_1 \text{ and } \cdots \text{ and } p_n)$, usually

$p_t = (p_{i_1} \text{ and } \cdots \text{ and } p_{i_m})$ for $\{i_1, \cdots, i_m\} \subseteq \{1, \cdots, n\}$, and (1) is directly obtained and no need to use Generate&Check-S.

The condition (3) of [Invariant Lemma] can be verified by using Generate&Check-T1 or T2 with $q_{tr}(s, s')$ defined as follows.

\[
(3) \quad q_{tr}(s, s') = (inv(s) \text{ implies } inv(s'))
\]
Invariants are fundamentally important properties of transition systems. They are asserting that something bad will not happen (i.e. safety property). However, it is sometimes also important to assert that something good will surely happen (i.e. liveness property). A (p leads-to q) property is a liveness property.

The (p leads-to q) property is adopted from the UNITY logic, the following definition is, however, not the same as the original one. In the UNITY logic, the basic model is the parallel program with parallel assignments, and (p leads-to q) is defined through applications of inference rules.
[Definition of (p leads-to q)] Let $TS = (St, Tr, In)$ be a transition system, let $Rst$ be the set of reachable states of $TS$, let $Tseq$ be the set of transition sequences of $TS$, and let $p, q$ be predicates with arity $(St, Data)$ of $TS$, where $Data$ is a data sort needed to specify $p, q$. Then $(p$ leads-to $q)$ is defined to be valid for $TS$ iff the following holds. Where $St^+$ denotes the set of state sequences with length more than zero, and $s \in \alpha$ means that $s$ is an element in $\alpha$ for $\alpha \in St^+$.

$$(\forall s\alpha \in Tseq)(\forall d \in Data)$$

$$(((s \in Rst) \text{ and } p(s, d) \text{ and } (\forall s' \in s\alpha)(\text{not } q(s', d))))$$

implies

$$(\exists \beta t \in St^+)(q(t, d) \text{ and } s\alpha\beta t \in Tseq))$$

It means that the system will get into a state $t$ with $q(t, d)$ from a state $s$ with $p(s, d)$ no matter what transition sequence is taken.
[(p leads-to q) Lemma] (Slide 2)

(1) \((\forall (\hat{s}, \hat{s}') \in \hat{Tr}) ((inv(\hat{s}) \text{ and } p(\hat{s}) \text{ and } (\text{not } q(\hat{s}))) \implies (p(\hat{s}') \text{ or } q(\hat{s}'))))\)

(2) \((\forall (\hat{s}, s') \in \hat{Tr}) ((inv(\hat{s}) \text{ and } p(\hat{s}) \text{ and } (\text{not } q(\hat{s}))) \implies (m(\hat{s}) > m(s')))\)

(3) \((\forall \hat{s} \in \hat{St}) ((inv(\hat{s}) \text{ and } p(\hat{s}) \text{ and } (\text{not } q(\hat{s}))) \implies (\exists \hat{s}' \in \hat{St})((\hat{s}, \hat{s}') \in \hat{Tr}))\)

(4) \((\forall \hat{s} \in \hat{St}) ((inv(\hat{s}) \text{ and } (p(\hat{s}) \text{ or } q(\hat{s})) \text{ and } (m(\hat{s}) = 0))) \implies q(\hat{s}))\)
The condition (1) and (2) of [(p leads-to q) Lemma] can be verified by using Generate&Check-T1 or T2 with $q_{tr}(s,s')$ defined as follows.

Note that $\hat{s}$s are omitted.

(1) $\quad q_{tr}(s,s') = ((\text{inv}(s) \text{ and } p(s) \text{ and } \neg q(s))) \implies (p(s') \text{ or } q(s'))$

(2) $\quad q_{tr}(s,s') = ((\text{inv}(s) \text{ and } p(s) \text{ and } \neg q(s))) \implies (m(s) > m(s'))$
The condition (3) and (4) of [(p leads-to q) Lemma] can be verified by using Generate&Check-S with \( p_{st}(s) \) defined as follows.

\[
\begin{align*}
(3) \quad p_{st}(s) &= ((\text{inv}(s) \text{ and } p(s) \text{ and } \neg q(s))) \\
&\quad \text{implies } (s =(*,1)=+ \text{ SS:State})
\end{align*}
\]

\[
\begin{align*}
(4) \quad p_{st}(s) &= ((\text{inv}(s) \text{ and } (p(s) \text{ or } q(s)) \text{ and } (m(s) = 0))) \\
&\quad \text{implies } q(s)
\end{align*}
\]

Note that \( (s =(*,1)=+ \text{ SS:State}) \) is a built-in search predicate that returns true if there exits \( s' \in St \) such that \( (s, s') \in Tr \) (i.e. \( (\exists s' \in St)((s, s') \in Tr)) \).
Searches on Time versus Space
There are recent attempts to extend the model checking with Maude for verifying infinite state transition systems. They are based on narrowing with unification, whereas the generate & check method is based on cover sets with ordinary matching and reduction.

Once a state configuration is properly designed, large number of patterns (i.e. elements of a cover set) that cover all possible cases are generated and checked easily, and it is an important future issue to construct proof scores for important problems/systems of significant sizes and do experiments for developing practical methods to obtain effective cover sets.

Module expressions of CafeOBJ is powerful, and are effective for constructing large specifications/proof-scores with systematic structures.