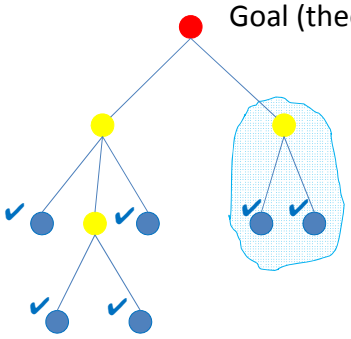
~~Model Checking: Basics~~
Proof Script Generation from Proof Scores (1)

Kazuhiro Ogata (JAIST, Japan)

i628e information processing theory

Motivation

Proof tree A proof can be regarded as a tree.



An instance of a proof rule

$$\frac{B \quad C}{A}$$

For example,

$$\frac{p(0) \quad (\forall x\colon Nat)\,\big(p(x) \Rightarrow p(x+1)\big)}{(\forall x\colon Nat)\,p(x)}$$

Motivation

$(\forall x\colon \mathrm{NAT})\ x + 0 = x$

$$\frac{p(0) \qquad (\forall x\colon Nat)\ \big(p(x) \Rightarrow p(x+1)\big)}{(\forall x\colon Nat)\ p(x)}$$

$0 + 0 = 0$ ✔
by axiom

$(\mathrm{c} + 0 = \mathrm{c}) \Rightarrow$
$(s(\mathrm{c}) + 0 = s(\mathrm{c}))$

A is used as axiom.

$$\frac{\begin{matrix}B\\ \vdots\end{matrix}}{A \Rightarrow B}$$

Note that c is an arbitrary Nat.

✔ $s(\mathrm{c}) + 0 = s(\mathrm{c})$
by axiom

axiom

$(\forall y\colon \mathrm{NAT})\ 0 + y = y$
$(\forall x, y\colon \mathrm{NAT})\ s(x) + y = s(x + y)$

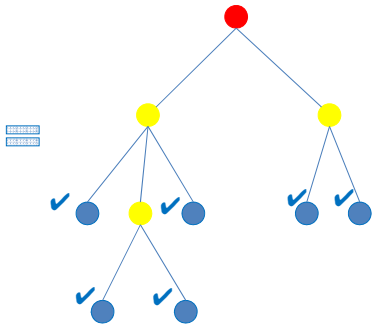
Motivation

Proof assistant

A sequence of commands (or tactics), or a *proof script*

Pros : it guarantees that once a proof is done, it is correct.

Pros : less flexible



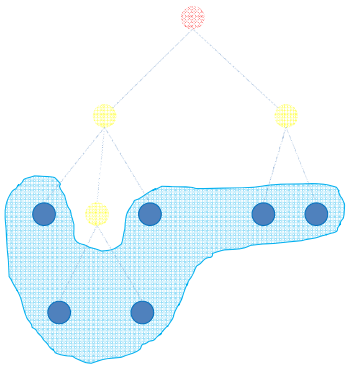
Motivation

Proof scores Proofs (or proof plans) written in an alg. spec. lang.

Pros : flexible, can be written as programs, etc.

Cons : subject to human errors

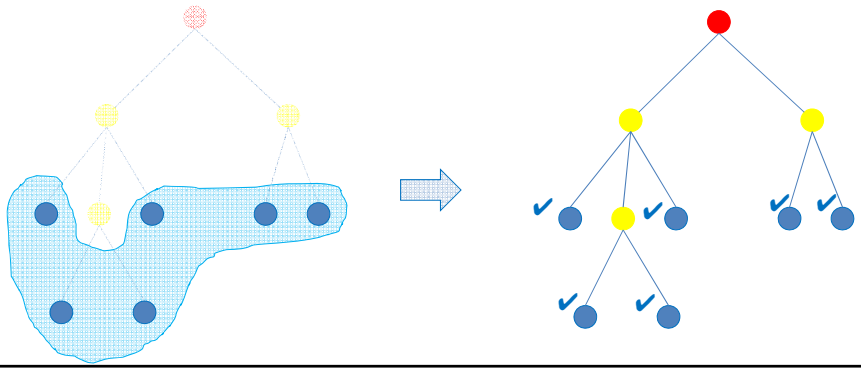
Only consisting of the leaf nodes of a proof tree



Motivation

Would like to check if proof scores are correct

One possible way to do so is to construct (or infer) proof scripts from proof scores



Outline

lecture 1

1. CafeOBJ, Maude & CafeInMaude
2. CafeOBJ
3. A running example
4. State machine & invariant
5. Formal specification
6. Proof scores
7. Summary

lecture 2

1. Review of lecture 1
2. CiMPA – CafeInMaude Proof Assistant
3. CiMPG – CafeInMaude Proof Generator
4. Summary & future directions

CafeOBJ, Maude & CafeInMaude

CafeOBJ & Maude are direct successors of OBJ3, the most famous alg. spec. lang. & sibling langs

Joseph Goguen
PI of OBJ3

Jose Meseguer
PI of Maude

Kokichi Futatsugi
PI of CafeOBJ

Adrian Riesco
Implementer of
CafeInMaude

CafeInMaude is the world 2nd implementation of CafeOBJ in Maude.

CafeOBJ

Queues are first-in & first-out (FIFO) data structures (collections) and defined as follows:

(1) empty is a queue (the empty queue).
(2) If e is an element and q is a queue, then $e \mid q$ is a queue, where e is the top of the queue.

Let e1, e2, e3 be elements. Some examples of queues are

e1 | empty

e1 | e2 | empty

e1 | e2 | e3 | empty

CafeOBJ

Queues are specified in CafeOBJ as follows:

```
mod! QUEUE(E :: TRIVerr) {
  [EQueue NeQueue < Queue]
  op empty : -> EQueue {constr}
  op _ | _ : Elt.E Queue -> NeQueue {constr}
  op enq : Queue Elt.E -> NeQueue
  op deq : Queue -> Queue
  op top : EQueue -> ErrElt.E
  op top : NeQueue -> Elt.E
  op top : Queue -> Elt&Err.E
  op _ \in _ : Elt.E Queue -> Bool .
  var Q : Queue  vars X Y : Elt.E
```

CafeOBJ

```
  eq enq(empty,X) = X | empty .
  eq enq(Y | Q,X) = Y | enq(Q,X) .
  eq deq(empty) = empty .
  eq deq(X | Q) = Q .
  eq top(empty) = err .
  eq top(X | Q) = X .
  eq X \in empty = false .
  eq X \in (Y | Q) = (if X = Y then true else X \in Q fi) .
}
```

Equations are used as left-to-right rewrite rules:

enq(empty,X) → X | empty -- (enq1)
enq(Y | Q,X) → Y | enq(Q,X) -- (enq2)

CafeOBJ

enq(enq(enq(empty,e1),e2),e3) is rewritten (or computed) as follows:

enq(enq(enq(empty,e1),e2),e3) by (enq1)
→ enq(enq(e1 | empty,e2),e3) by (enq2)
→ enq(e1 | enq(empty,e2),e3) by (enq1)
→ enq(e1 | e2 | empty,e3) by (enq2)
→ e1 | enq(e2 | empty,e3) by (enq2)
→ e1 | e2 | enq(empty,e3) by (enq1)
→ e1 | e2 | e3 | empty

enq(empty,X) → X | empty -- (enq1)
enq(Y | Q,X) → Y | enq(Q,X) -- (enq2)

CafeOBJ

CafeOBJ can do symbolic execution in a sense, which makes it possible to do theorem proving.

Let us prove $(\forall e\colon \mathrm{Elt.\,E})(\forall q\colon \mathrm{Queue})\ \mathrm{top}(e \mid q) = e$.

```
open QUEUE .
op e : -> Elt.E .
op q : -> Queue .
red top(e | q) = e .
close
```

✔

e denotes an arbitrary element.
q denotes an arbitrary queue.

CafeOBJ

Let us prove $(\forall x, y\colon \mathrm{Elt.\,E})(\forall q\colon \mathrm{Queue})$
$x \in \mathrm{enq}(q, y) = (\text{if } x = y \text{ then true else } x \in q \text{ fi})$

It is proved by structural induction on q + case splitting.

```
open QUEUE .
ops x y : -> Elt.E .
red x \in enq(empty,y) = (if x = y then true else x \in empty fi) .
close
```

✔

```
open QUEUE .
ops x y z : -> Elt.E .
op q : -> Queue .
eq X:Elt.E \in enq(q,Y:Elt.E) = (if X = Y then true else X \in q fi) .
red x \in enq(z | q,y) = (if x = y then true else x \in z | q fi) .
close
```

Not yet

CafeOBJ

```
open QUEUE .        ✔
ops x y z : -> Elt.E .
op q : -> Queue .
eq X:Elt.E \in enq(q,Y:Elt.E) = (if X = Y then true else X \in q fi) .
eq x = z .  eq y = z .
red x \in enq(z | q,y) = (if x = y then true else x \in z | q fi) .
close
```

```
open QUEUE .        ✔
ops x y z : -> Elt.E .
op q : -> Queue .
eq X:Elt.E \in enq(q,Y:Elt.E) = (if X = Y then true else X \in q fi) .
eq x = z .  eq (y = z) = false .
red x \in enq(z | q,y) = (if x = y then true else x \in z | q fi) .
close
```


CafeOBJ

```
open QUEUE .        ✔
ops x y z : -> Elt.E .
op q : -> Queue .
eq X:Elt.E \in enq(q,Y:Elt.E) = (if X = Y then true else X \in q fi) .
eq (x = z) = false .  eq y = z .
red x \in enq(z | q,y) = (if x = y then true else x \in z | q fi) .
close
```

```
open QUEUE .        ✔
ops x y z : -> Elt.E .
op q : -> Queue .
eq X:Elt.E \in enq(q,Y:Elt.E) = (if X = Y then true else X \in q fi) .
eq (x = z) = false .  eq (y = z) = false .  eq x = y .
red x \in enq(z | q,y) = (if x = y then true else x \in z | q fi) .
close
```

CafeOBJ

```
open QUEUE .        ✔
ops x y z : -> Elt.E .
op q : -> Queue .
eq X:Elt.E \in enq(q,Y:Elt.E) = (if X = Y then true else X \in q fi) .
eq (x = z) = false .  eq (y = z) = false .  eq (x = y) = false .
red x \in enq(z | q,y) = (if x = y then true else x \in z | q fi) .
close
```

Each open-close fragment used in a proof score corresponds to one leaf node in a proof tree.

Even if some open-close fragments are missing, CafeOBJ complains at all.

A running example

A mutual exclusion protocol called Qlock, an abstract version of the Dijkstra binary semaphore.

Pseudo code for process *p*:

Loop "Remainder Section"
  rs: enqueue(*queue*,*p*);
  ws: repeat until top(*queue*) = *p*;
    "Critical Section"
  cs: dequeue(*queue*);

Each process *p* is located at rs, ws or cs.

Initially, *p* is located at rs & *queue* is empty.

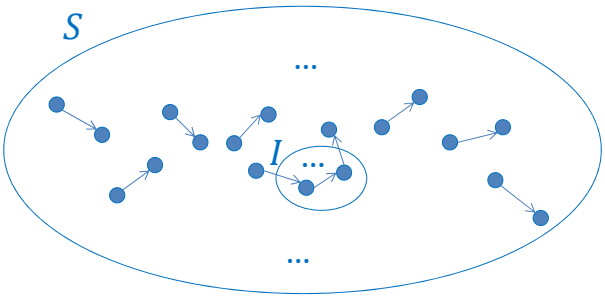
State machine & invariant

The set $I \subseteq S$ of initial states

State machine $M \triangleq \langle S, I, T \rangle$

A set of states

A binary relation $T \subseteq S \times S$ over states

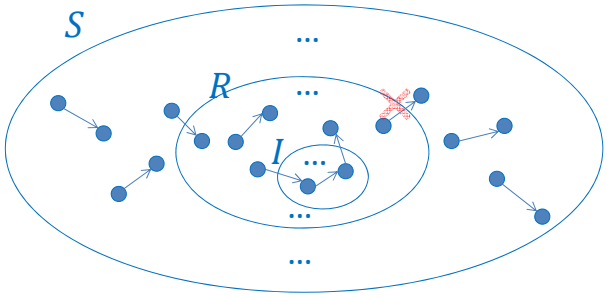


State machine & invariant

The set R of reachable states wrt M

(1) $I \subseteq R$

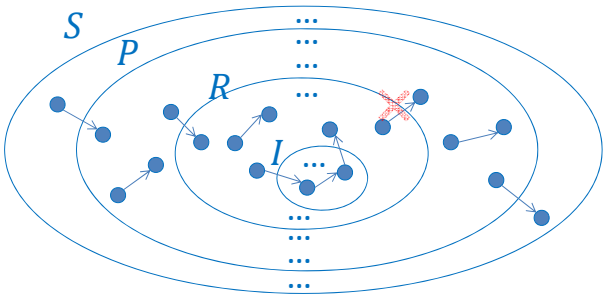
(2) If $s \in R$ & $(s, s') \in T$, then $s' \in R$.



State machine & invariant

A state predicate $p : S \to \text{Bool}$ is invariant wrt M iff $(\forall s \in R) p(s)$.

$p : S \to \text{Bool}$ can be interpreted as the set P of states s.t. $\{s \in R \mid p(s)\}$ and vice versa.

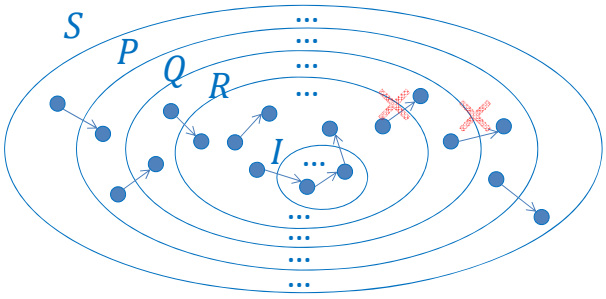


State machine & invariant

An inductive invariant is a state predicate $q : S \to \text{Bool}$ that satisfies the following:

(1) $(\forall s \in I)\ q(s)$

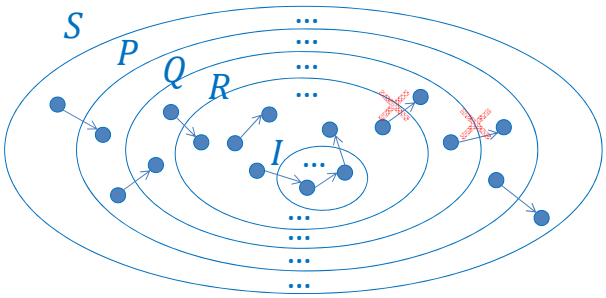
(2) $(\forall (s, s') \in T)\ (q(s) \Rightarrow q(s'))$



State machine & invariant

To prove that $p : S \to \mathrm{Bool}$ is invariant wrt M, we need to find an inductive invariant $q : S \to \mathrm{Bool}$ s.t. $(\forall s \in S)\ (q(s) \Rightarrow p(s))$.

Such q or part of such q is called a *lemma* of the proof.



Formal specification

Treating states as black boxes but possible to observe values from them. Given a state s & a process ID i,

$\mathrm{queue}(s)$	the content of $queue$ used in Qlock
$\mathrm{pc}(s, i)$	the location where process i is located

queue & pc are called *observer functions*.

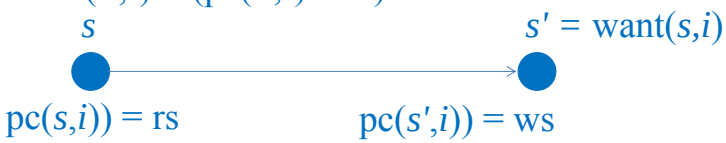
Let init be an arbitrary initial state.

$\mathrm{eq}\ \mathrm{queue}(\mathrm{init}) = \mathrm{empty}$.
$\mathrm{eq}\ \mathrm{pc}(\mathrm{init}, I) = \mathrm{rs}$.

Formal specification

```
ceq pc(want(S,I),J)
  = (if I = J then ws else pc(S,J) fi)        if c-want(S,I) .
ceq queue(want(S,I)) = enq(queue(S),I)        if c-want(S,I) .
ceq want(S,I)        = S                      if not c-want(S,I) .
```

where c-want(S,I) = (pc(S,I) = rs)



pc(s′,j) = pc(s,j) **if** $i \neq j$
queue(s′) = enq(queue(s),i)

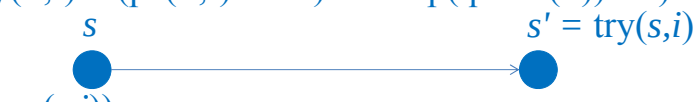
Loop "Remainder Section"
rs: enqueue(*queue*);
ws: repeat until top(*queue*) = *p*;
"Critical Section"
cs: dequeue(*queue*);

want is called a *transition functions*.

Formal specification

```
ceq pc(try(S,I),J)
= (if I = J then cs else pc(S,J) fi)    if c-try(S,I) .
eq  queue(try(S,I)) = queue(S) .
ceq try(S,I)        = S                 if not c-try(S,I) .
```

where c-try(S,I) = (pc(S,I) = ws) and top(queue(S)) = I)



pc(s′,j) = pc(s,j) **if** $i \neq j$
queue(s′) = queue(s)

Loop "Remainder Section"
rs: enqueue(*queue*);
ws: repeat until top(*queue*) = *p*;
"Critical Section"
cs: dequeue(*queue*);

try is called a *transition functions*.

Formal specification

```
ceq pc(exit(S,I),J)
  = (if I = J then rs else pc(S,J) fi)        if c-exit(S,I) .
ceq queue(exit(S,I)) = deq(queue(S))          if c-exit(S,I) .
ceq exit(S,I)        = S                      if not c-exit(S,I) .
```

where c-exit(S,I) = (pc(S,I) = cs)

$pc(s',j) = pc(s,j)$ **if** $i \neq j$
$queue(s') = deq(queue(s))$

```
Loop "Remainder Section"
  rs: enqueue(queue);
  ws: repeat until top(queue) = p;
      "Critical Section"
  cs: dequeue(queue);
```

exit is called a *transition functions*.

Formal specification

Mutual exclusion property: there is always at most one process in the critical section.

```
eq inv1(S:Sys,I:Pid,J:Pid)
  = (((pc(S,I) = cs) and pc(S,J) = cs) implies I = J) .
```

To verify that Qlock enjoys the property, it suffices to prove $(\forall s : Sys)(\forall i, j: Pid)\ inv1(s, i, j)$.

Sys is a sort (or type) of the reachable states of the state machine formalizing Qlock & Pid is a sort (or a type) of process IDs.

Proof score

To prove $(\forall s : \mathrm{Sys})(\forall \mathrm{i}, \mathrm{j}: \mathrm{Pid})\ \mathrm{inv1}(\mathrm{s}, \mathrm{i}, \mathrm{j})$, we first use structural induction on s, Theorem of Constants and (an instance of) the induction hypothesis.

✔

```
open QLOCK .
ops i j : -> Pid .
red inv1(init,i,j) .
close
```

Not yet

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```


Proof score

Not yet

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

Not yet

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
red inv1(s,i,j) implies inv1(exit(s,k),i,j) .
close
```

Proof score

To discharge the three cases (try, want & exit), we use case splitting by adding some equations.

Based on the condition of each transition function, the result returned by CafeOBJ for each open-close fragment, etc., we find equations to do case splitting.

Proof score

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = rs .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

Not yet

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq (pc(s,k) = rs) = false .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

✔

Proof score

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = rs .  eq i = k .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

✔

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = rs .  eq (i = k) = false .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

Not yet

Proof score

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = rs .  eq (i = k) = false .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

CafeOBJ returns the following for the open-close fragment:

```
((((if (k = j) then ws else pc(s,j) fi) = cs) and ((i = j) and (pc(s,i) =
cs))) xor (((pc(s,j) = cs) and ((pc(s,i) = cs) and ((i = j) and ((if (k = j)
then ws else pc(s,j) fi) = cs)))) xor (true xor (((pc(s,j) = cs) and
((pc(s,i) = cs) and ((if (k = j) then ws else pc(s,j) fi) = cs))) xor
((pc(s,i) = cs) and ((if (k = j) then ws else pc(s,j) fi) = cs))))))
```

Proof score

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = rs .  eq (i = k) = false .  eq j = k .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

✔

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = rs .  eq (i = k) = false .  eq (j = k) = false .
red inv1(s,i,j) implies inv1(want(s,k),i,j) .
close
```

✔

Proof score

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq pc(s,k) = ws .   eq top(queue(s)) = k .
eq i = k .   eq (j = k) = false .   eq pc(s,j) = cs .
red inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

Not yet

Based on the three equations, we can conjecture the lemma:

```
eq inv2(S:Sys,I:Pid) = (pc(S,I) = cs implies top(queue(S)) = I) .
```

Proof score

```
open QLOCK .
op s : -> Sys .
ops i j k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq [:nonexec] : inv2(s,I:Pid) = true .
eq pc(s,k) = ws .   eq top(queue(s)) = k .
eq i = k .   eq (j = k) = false .   eq pc(s,j) = cs .
red inv2(s,j) implies inv1(s,i,j) implies inv1(try(s,k),i,j) .
close
```

✔

Proof score

```
open QLOCK .
op s : -> Sys .
ops i k : -> Pid .
eq [:nonexec] : inv1(s,I:Pid,J:Pid) = true .
eq [:nonexec] : inv2(s,I:Pid) = true .
eq pc(s,k) = cs .   eq (i = k) = false .   eq pc(s,i) = cs .
red inv1(s,i,k) implies inv2(s,i) implies inv2(exit(s,k),i) .
close
```

✔

$(\forall s : \mathrm{Sys})(\forall \mathrm{i, j: Pid})\ \mathrm{inv1(s, i, j)}$ &
$(\forall s : \mathrm{Sys})(\forall \mathrm{i: Pid})\ \mathrm{inv2(s, i)}$ are proved by simultaneous (structural) induction on s.

Summary

Queues have been used as an example to describe formal (data) specifications in CafeOBJ and proof scores for data.

Qlock has been used as an example to describe formal (systems) specifications in CafeOBJ and proof scores for systems.

Even if some open-close fragments, leaf nodes in a proof tree, are missing, CafeOBJ complains at all.
Thus, proof scores are subject to human errors.

Summary

Some case studies have been done in which some security protocols are formally verified with CafeOBJ and poof scores.

e-commerce protocols, such as iKP and SET

Kazuhiro Ogata, Kokichi Futatsugi: Proof Score Approach to Analysis of Electronic Commerce Protocols. International Journal of Software Engineering and Knowledge Engineering 20(2): 253-287 (2010)

authentication protocols, such as NSLPK and TESLA

Kazuhiro Ogata, Kokichi Futatsugi:Proof Scores in the OTS/CafeOBJ Method. FMOODS 2003: 170-184

Iakovos Ouranos, Kazuhiro Ogata, Petros S. Stefaneas: TESLA Source Authentication Protocol Verification Experiment in the Timed OTS/CafeOBJ Method: Experiences and Lessons Learned. IEICE Transactions 97-D(5): 1160-1170 (2014)