# Machine-checked correctness of a secure information flow analyzer (preliminary report)

David A. Naumann

Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, 07030, USA
naumann@cs.stevens-tech.edu

Department of Computer Science
Technical Report CS-2004-10
August 24, 2004

# Machine-checked correctness of a secure information flow analyzer (preliminary report)

David A. Naumann*

August 24, 2004

## 1 Introduction

This document reports on the use of the PVS theorem prover [ORS92] (version 3.1) to

- formalize a semantic model for a core fragment of Java; and
- use the model to machine check the soundness of a secure information flow analysis.

Soundness means that a program deemed "safe" by the static analysis, relative to a given policy, does satisfy the noninterference property for that policy. The analysis and noninterference proof are based on the work of Banerjee and Naumann [BN03a], which handles essentially the sequential core of Java including also the code-based access control mechanism [Gon99]. Noninterference is defined in terms of a denotational semantics which is being used in other work on program verification.

The PVS verification has been completed as described herein.[1] But some revision is planned both to polish the work and to eliminate workarounds currently needed due to minor bugs in the PVS system. In its current form, this technical report has little expository material; for that, the reader is advised to consult [BN03a]. In lieu of exposition there are some working notes *note:[indicated like this one]*. A revised version of the tech report will be released in due course, whence the title "preliminary".

The encoding of the language is a deep embedding. The encoding follows the definitions in the paper [BN03a] very closely, using the predicate types and dependent types of PVS extensively. The use of dependent predicate types, e.g., to express the absence of dangling pointers or ill-typed states, means that type-correctness of our definitions requires proof in some cases. (One consequence is type soundness of the language, see subsection 2.2.)

Axioms are used only for two purposes: (1) To express the assumption that a given program is syntactically well formed (e.g., subclassing is acyclic) and type correct. (2) To express the assumption that the memory allocator, otherwise arbitrary, returns fresh locations. The axioms are simple and soundness is obvious. *note:[PVS theory interpretations have been given to justify soundness,*

*Supported in part by the National Science Foundation under grants CCR-0208984 and CCR-ITR-0326540, the Office of Naval Research under grant N00014-01-1-0837, and the NJ Commission on Science and Technology.

[1] Available from the author, CVS release tag accinf-1-0. A PVS patch from Sam Owre is required; it is included in the pvs-strategies file.

*but as explained in the sequel (Subsection 2.3) these interpretations are slightly simplified as a workaround for unresolved PVS problems (or my ineptitude).]*

Here are the difference between the current PVS development and [BN03a].

- Runtime access control (stack inspection) is omitted from the language and permission-dependency is omitted from security policies. Security policy takes the usual form of labels, as in, e.g., [VSI96, BN02].
- Instead of the specific two-element lattice $\{high, low\}$ used in the paper, security levels are in an arbitrary lattice.
- In our papers, the analysis is specified as an inductively defined relation "command $S$ is secure and writes fields at least $\kappa_0$ and locals at least $\kappa_1$". Here, the analysis is given as a function that takes $S$, together with a fixed policy (labelling) for variables and fields; it returns a pair $(\kappa_0, \kappa_1)$ which is in some sense the most precise ones for which the relation holds. Similarly, there is a function that gives the read-level for an expression, relative to a given policy for variables and fields.
- To simplify the formalization slightly, each method has exactly one parameter and one local variable; the generalization is straightforword.

To complete the project, access control, and permission-dependent policy, will be added following [BN03a]. Many of the proofs reflect that this was my first project using PVS; they deserve to be polished to fully exploit the capabilities of PVS.

This is not the first machine checked proof for soundness an information flow analysis. Strecker [Str03] uses Isabelle/HOL to show soundness for a language and security typing rules very similar to those in [BN02], i.e., a sequential fragment of Java, without access control. Strecker's result, like that in [BN02], makes an assumption of "parametricity" for the memory allocator, in order to use equality rather than an arbitrary bijection on locations. This simplifies the proofs considerably, but is not realistic in that the address chosen by a memory allocator depends on all objects currently allocated. Strecker also confines attention to the two-element lattice.

The present work confirms the wisdom of those simplifications. Although an arbitrary lattice is needed for practical applications, it results in an additional quantifier in each of the key properties (indistinguishability, write-confinement, and noninterference) and this pervades the proofs. Treating memory allocation realistically necessitates the use of a bijection on locations in the definition of indistinguishability (on which the definitions of write-confinement and noninterference are based). This was not terribly daunting in the step from [BN03b] to [BN03a], where the proofs are worked out in some detail. But as Strecker [Str03] remarks, it means that there are fewer opportunities to exploit provers' built-in equality reasoning. In the present work, various proofs involve manipulation of composed binary relations, but this has not proved to be terribly onerous.

The effort uncovered a minor bug in [BN03b]: We had neglected to impose on the semantic domains that the domain of the heap never shrinks, but this was assumed implicitly in some proofs. To my suprise this was the only such bug. This is not to say no mistakes were made during the PVS development: In the generalization to an arbitrary lattice of levels, I formulated revised definitions off the top of my head, without proving correctness on paper. The aim was to find out whether PVS could serve as a proof assistant and help find the correct formulations. I was not disappointed.

**Outline:** Section 2 is an overview of the theories. Section 3 gives the PVS theories. For the proofs, one must consult the PVS files which are available from the author on request. Section 4 gives the transcript showing that all proof obligations are discharged.

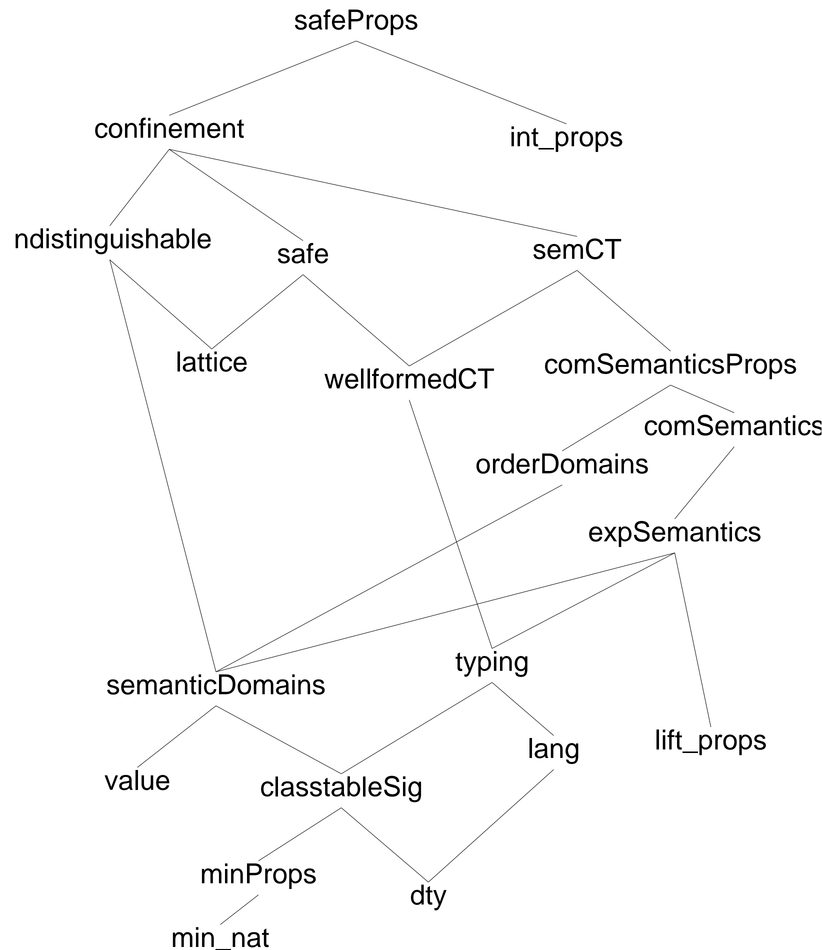


Figure 1: The import hierarchy.

# 2 Overview

## 2.1 Outline of theories

We give brief descriptions of the PVS theories formalizing the general theory of [BN03b]. Figure 1 depicts the import relation on these theories. Section 2.3 describes additional theories that instantiate the general ones.

The following PVS theories formalize the syntax of the programming language. Commands are formalized as an inductive datatype. A well formed program is given in the form of a class table [IPW01], i.e., a collection of class declarations, each giving a superclass, field types, method signatures, and method implementations. We use one theory to formalize the method implementations and another to formalize all other elements of a well formed class table.

- *dty* —data type names
- *lang* —expressions (PVS datatype *exp*) and commands (PVS datatype *com*)

  These are annotated with types, i.e., in the form of a typechecked abstract syntax tree; this makes for simple typing rules.

- *classtableSig* —field and method signatures; variable context for typing judgements; subtyping and inheritance; assumption that subtyping is well founded and that method types are invariant w.r.t. subclassing
- *typing* —rules for typing expressions and commands, in the context of a given *classtableSig*
- *wellformedCT* —well formed class table for a given *classtableSig*; assumption that method bodies are present (where stipulated by the signature) and typable

The following theories formalize a denotational semantics for the language. Briefly: a command denotes a function mapping initial state to final state, where a state consists of a heap and a *store* (that assigns values to local variables in scope). A method denotes a function that sends a state (where the store holds initial parameter values and the target object) to a result consisting of updated heap and result value. A method environment provides meanings for all declared methods. It is a parameter of the semantic function for commands and is defined as a limit of approximations.

- *value* —the PVS datatype of primitive values (literals and locations)
- *semanticDomains* —meanings of program datatypes, expressions, commands, methods, and method environments. Imports *value*, *classtableSig*.
- *expSemantics* —semantics of typable expressions. Imports typing and semanticDomains.
- *comSemantics* —semantics of typable commands; assumption that memory allocator returns fresh addresses
- *orderDomains* —orderings on semantic domains, to justify fixpoint definitions, and concrete characterizations to facilitate proofs
- *comSemanticsProps* —monotonicity of command constructs and methods, to justify fixpoint definitions
- *semCT* —semantics of method declaration; semantics of a wellformed classtable (limit of chain of approximate method environments). Imports *wellformedCT*, *orderDomains*, and *comSemanticsProps*.

The static analysis for secure information flow is formalized by theory

- *safe* —security labels/policy; assumption that method policy is invariant w.r.t. subclassing; static analysis for exp, com, and methods. Imports lattice and wellformedCT.

The following theories formalize the semantic properties including noninterference, and prove the main results.

- *indistinguishable* —L-indistinguishable states etc; read and write confinement. Imports *semanticDomains*, *lattice*.
- *confinement* —a safe expression is read-confined; a safe command is write-confined; semantics of classtable is write-confined. Imports *indistinguishable*, *semCT*, *safe*.
- *safeProps* —main results: safe commands and CT are noninterfering with respect to the policy given in *safe*

The following theories provide convenient definitions and properties of standard mathematical notions not found in the standard PVS 3.1 prelude. Presumably these results can and should be found in existing libraries but I chose to do the initial versions myself as exercises.

- *lift_props* —properties of lifting
- *int_props* —properties of the binary *max* operator on integers
- *minProps* —properties of the *min* operator on sets of integers
- *lattice* —definitions and basic properties for lattices

As discussed in subsection 2.3, axioms are used only to express assumptions like "the program is well formed" (mentioned in the above descriptions of *classtableSig* and *wellformedCT*). To justify soundness of these axioms —and to illustrate how the encoding represents Java source code— the theories discussed in that subsection provide theory interpretations.

## 2.2 On type soundness for the Java fragment.

The semantics is defined for typable programs and type soundness follows from the semantics being well defined. In the paper [BN03b] we do not prove in detail that the semantics is well defined, whereas that *is* done in the PVS version. In order to use the built-in induction schemes of PVS for algebraic datatypes with subtypes, the semantic definitions are extended here to include untypable commands and expressions: these denote error/abort. The constituents of a typable command are typable, so the extended part of the definition is not relevant to the semantics of a typable command; thus the machine checked proof shows type soundness and confirms well definedness of the semantics in the paper.

This is perhaps not of major interest in itself, though it shows that denotational proofs of type soundness can be as tractable as proofs based on operational semantics. It may be of more interest with the planned extension of the language to include generics. *note:[And for that purpose it might be more clear to use an induction scheme tailored for the datatype, so the semantic definitions can be given exactly for the typable phrases.]*

## 2.3 On the use of axioms

The formalization is essentially definitional. Axioms are used in a limited way to express assumptions in generic theories. This is simpler than using the parameterization/assumption mechanism in PVS, though it amounts to the same thing. We give interpretations for these theories, to show soundness and to illustrate how Java source code is modeled.

**Program syntax axioms** The most substantial assumptions are for program syntax. They are easily discharged for (the encoding of) a program that is well formed according to the rules of Java [AG98].

Theory *classtableSig* defines the signature of a class table –i.e., closed collection of classes including the root class Object– as follows: nonempty sets *Classnames*, *Varnames*, and *Methnames* are assumed, as well as distinguished elements *Result* and *Self* in *Varnames*. The intended interpretation is as follows.

- *Classnames* is the set of declared classes, which includes the distinguished class *Object*.

- $Varnames$ is the set of names used for local variables, parameters, and fields; it includes at least the distinguished names $Self$ and $Result$.
- $Methnames$ is the set of names for methods.

The theory, *lang*, that gives concrete syntax is parameterized on these sets of names. Theory *classtableSig* also postulates the existence of functions:

- *super* - intended to map each class name to its direct superclass (and *Object* to itself)
- $ftype$ - intended to map each (C,f) to the type of field f in class C

  The "void" type *unitT* is used to encode that f is neither declared nor inherited in C.
- *mtype* - intended to map each (C,m) to the parameter and result types for method m declared or inherited in class C; bottom encodes that m is not defined in C.

The axioms of *classtableSig* merely express the usual syntactic conditions/assumptions on signatures in a wellformed program:

- *Result* and $Self$ are distinct names and may not be used as parameter names.
- The subclass relation determined by super is acyclic and has top element *Object*. This is expressed in a way that allows there to be infinitely many classes: for every $C$, there is $n$ such that $iterate(super, n)(C) = Object$.
- Typing is invariant: if field f or method m is defined in class C then its type in subclasses of C is the same as in C.

Theory $wellformedCT$ defines wellformedness conditions for method bodies. It postulates a function, *mbody*, that assigns to each C,m a suitable body for m (including local variables) with respect to the type given by the classtableSig. Axioms merely express that

- Every defined method has a body, either declared or inherited, and declared bodies are typable.
- Local variable names are distinct from *Result* and $Self$.

To demonstrate consistency and show the correspondence between concrete code and our model, we have included a theory that interprets *classtableSig* and $wellformedCT$ with a simple Java program.

*note:[These theory interpretations are incomplete due to some complications with the theory interpretation mechanism vis a vis dependent types and theory importation.]*

- *note:[Xwf_combined is a simple Java program encoded in our syntax, to give a theory interpretation for classtableSig and wellformedCT. In fact it gives an interpretation for Xcombined, described below. ]*
- *note:[wellformedSigCT merges classtableSig and wellformedCT, to avoid having to give a single theory interpretation rather than two hierarchical ones. This is a workaround to avoid problems with the PVS theory interpretation mechanism (under investigation). ]*
- *note:[Xcombined merges XclasstableSig and XwellformedCT, which are simplified from their counterparts to facilitate study of the problems.]*

*note:[In the first version of the development (CVS release accinf-1-0), function mbody in theory wellformedCT was postulated to have a dependent type that expresses part of the conditions mentioned above. However, this type involves function DefinedMeth from theory classtableSig. At this time I have been unable to express the theory interpretation in a way accepted by PVS. Pending resolution of this problem, I have reformulated wellformedCT slightly, so that the postulated function is mbody-raw and mbody is* defined *from mbody-raw; its type is then a consequence of explicitly stated axioms. ]*

*note:[In fact there seem to be problems in the theory interpretation mechanism that prevents two-step interpretation of classtableSig and then wellformedCT. Pending resolution of this problem, I have merged these two theories into one called wellformedSigCT, and it is the latter for which an interpretation has been given. ]*

**Semantic axioms** The semantics is parameterized on the type, *Loc*, of memory locations and two functions. The postulated function *loctype* assigns a class name to every location, allocated or not. The memory allocator function, $fresh$, is postulated to return an unallocated location of designated type; this is the only axiom.

Theory $heapModel$ gives an obvious model in which a location is a pair $(n, C)$ with $n$ a natural number; $fresh(C, h)$ returns $(n, C)$ where $n$ is the least number that is not allocated in heap $h$. Typical implementations treat $h$ as a mapping *nat* → *obstate* where an *obstate* maps fields to values and the object type is a special field. Given that the type field is immutable, it is isomorphic to curry this function to get *nat* × Classnames → *obstate* which is our model.

*note:[The theory interpretation by which heapModel shows consistency has not been completed. It needs to interpret both semanticDomains (Loc, loctype) and comSemantics (fresh) and should pose no difficulty once the above-mentioned problems with theory interpretation are resolved.]*

**Security policy axioms** Theory $safe$ declares the information flow policy using functions $flowPolicy$ and $flowPolicyFld$ that correspond to the ordinary signature functions *mtype* and $ftype$. The intended interpretation is that $flowPolicyFld(C)(f)$ is the level of field $f$ in class $C$ and $flowPolicy(C, m)$ is a tuple $(s, p, r, h)$ where $s$ (respectively $p$, $r$, $h$) is an upper bound on the level of the target object (resp. $p$ an upper bound on parameter level, $r$ an upper bound on the result value, and $h$ a lower bound on levels of fields written). An axiom expresses the assumption that policy is invariant with respect to subtyping. (This is needed for the same reason it is needed for ordinary types: so that uses can be checked in terms of static types.)

# 3 PVS code

The theories appear in the order listed above. *note:[Need to update these and add the theory interpretation theories.]*

*note:[Due to a bug in PVS Latex-generation, some of the theories are given here in raw ascii form. This does have the virtue that comments don't get discarded. ]*

```
dty[Classname: TYPE+]: THEORY
 BEGIN

  dty: DATATYPE
   BEGIN
    unitT: unitT?
    boolT: boolT?
    classT(name: Classname): classT?
```

```
  END dty

 C, D: VAR Classname

 inj_classT: LEMMA classT(C) = classT(D) ⇒ C = D

END dty
```

```
lang[Classname: TYPE+, Varname: TYPE+, Methname: TYPE+]: THEORY
 BEGIN

  IMPORTING dty[Classname]

  exp: DATATYPE
   BEGIN
    vblV(name: Varname): vblV?
    boolV(val: bool): boolV?
    nullV: nullV?
    eqtest(ex1: exp, dt1: dty,
           ex2: exp, dt2:
           dty): eqtest?
    fieldAccess(ex: exp, dt: dty,
                f: Varname): faccess?
    typetest(ex: exp, extype: dty,
             checkedtype: dty): typetest?
    cast(ex: exp, extype: dty,
         casttotype: dty): cast?
   END exp

  com: DATATYPE
   BEGIN
    assign(vbl: Varname, ex: exp,
           extype: dty): assign?
    fieldUpdate(ex1: exp, f:
                Varname, ex2: exp,
                ex1ty: dty, ex2ty:
                dty): fieldUpdate?
    new(vbl: Varname, name:
        Classname): new?
    mcall(vbl: Varname, ob: exp,
          name: Methname, arg:
          exp, ex1ty: dty, ex2ty:
          dty): mcall?
    ifelse(ex: exp, s1: com, s2:
           com): ifelse?
    seq(fst: com, snd: com): seq?
    skip: skip?
   END com

 END lang
```

```
% $Revision: 1.3 $ $Date: 2004/05/04 04:46:13 $

classtableSig: THEORY
%-----------------------------------------
% Names; field & method types; subclassing
%-----------------------------------------
 BEGIN
  CONVERSION- TotalFun_to_LPartFun

  %------------------------------------
  % Names used in syntax and semantics
  %------------------------------------

  Classname: TYPE+  % The classes for which declarations exist.
  Object: Classname

  IMPORTING dty[Classname]

  Varname: TYPE+
  Self: Varname
  Result: Varname

  distinct_names: AXIOM
    Self /= Result

  Methname: TYPE+

  %--------------------------------------------------------
  % Assumptions and definitions about super and subclassing
  % (To avoid need for lifting, super(Object) = Object.)
  %--------------------------------------------------------

  super: [Classname -> Classname]

  C, D, E:   VAR Classname
  T, T1, T2: VAR dty
  i, j:      VAR nat

  super_fin_top: AXIOM EXISTS j: j > 0 & iterate(super,j)(C) = Object


% TBD want to allow rewriting of <= in this theory but nowhere else,
% and I'm not sure how to refer to it in AUTO_REWRITE- declaration
% without giving it a prefix name.  There's no other point for 'leqClassname';
% but if it's dropped, need to drop it's expansion from proof of object_top.

  leqClassname(C, D): bool = EXISTS j:  iterate(super,j)(C) = D
;
  <=(C, D): bool = leqClassname(C,D)
;
  <=(T1, T2): bool =
      ( classT?(T1) & classT?(T2) & name(T1) <= name(T2) )
      OR ( unitT?(T1) & unitT?(T2) ) OR ( boolT?(T1) & boolT?(T2) )
```

```
%--------------------------
% Depth of class and super
%--------------------------

IMPORTING minProps

stepsToObj(C)(j): bool  =  iterate(super,j)(C) = Object

% Local lemmas for proof of cdepth_super & tccs
stepsToObj_nonempty: LEMMA nonempty?(stepsToObj(C))
step_to_obj: LEMMA stepsToObj(C)(j+1) => stepsToObj(super(C))(j)
steps_to_obj_B: LEMMA FORALL j: j>0 & stepsToObj(C)(j) => stepsToObj(super(C))(j-1)
steps_to_obj_A: LEMMA C /= Object => NOT stepsToObj(C)(0)

cdepth(C): nat = min(stepsToObj(C))

cdepth_super: LEMMA C /= Object => cdepth(super(C)) < cdepth(C)

%-------------
% Subclassing
%-------------

subcl_reflexive: LEMMA C <= C
subcl_transitive: LEMMA C <= D & D <= E  =>  C <= E
subcl_reflexiveT: LEMMA T <= T
subcl_transitiveT: LEMMA T <= T1 & T1 <= T2  =>  T <= T2
object_top: LEMMA FORALL C: C <= Object
super_above: LEMMA C <= super(C)

below(T1)(T2): bool = T2 <= T1
Below(T1): TYPE = (below(T1))

class_below_class: JUDGEMENT Below(T1: (classT?)) SUBTYPE_OF (classT?)
class_below_class_lem: LEMMA classT?(T1) & T2 <= T1 => classT?(T2)

%-----------------------------
% Field and method signatures
%-----------------------------

ftype: [Classname, Varname -> dty]
mtype: [Classname, Methname
        -> lift[[# parN: Varname, parT: dty, resT: dty #]]]

n, f: VAR Varname
m: VAR Methname

inherit_fields: AXIOM
  C <= D & ftype(D,f) /= unitT => ftype(C,f) = ftype(D,f)

inherit_meths: AXIOM
  C <= D & up?(mtype(D,m)) => mtype(C,m) = mtype(D,m)
```

```
  fields_finite: AXIOM % I don't believe this is used anywhere
    is_finite( { f | ftype(C,f) /= unitT } )

  mtypes_par_not_SR: AXIOM
    up?(mtype(C,m)) => Let n = down(mtype(C,m))`parN IN n/=Self & n/=Result

  fields(C: Classname)(f: Varname): bool = ftype(C,f) /= unitT
  Fields(C): TYPE = (fields(C))

  definedMeth(C)(m): bool = up?(mtype(C,m))
  DefinedMeth(C): TYPE = (definedMeth(C))

  def_subcl: LEMMA
    definedMeth(D)(m) AND C<=D  =>  definedMeth(C)(m)

  def_super: LEMMA
    definedMeth(super(C))(m) => definedMeth(C)(m)

AUTO_REWRITE+ class_below_class_lem, def_subcl

  varnameNotSR(n): bool =    n /= Self & n /= Result % names allowed for locals/params
  VarnameNotSR: TYPE = (varnameNotSR)

  defined_mtype_par_not_SR: JUDGEMENT   % from axiom above
    mtype(C: Classname, m: DefinedMeth(C)) HAS_TYPE
     (up?[[# parN: VarnameNotSR, parT: dty, resT: dty #]])

  resType(C)(m: DefinedMeth(C)): dty =
   LET r: [# parN: Varname, parT: dty, resT: dty #]
     = down(mtype(C,m)) IN r`resT

  parType(C)(m: DefinedMeth(C)): dty =
   LET r: [# parN: Varname, parT: dty, resT: dty #]
     = down(mtype(C,m)) IN r`parT

  varNotSRpar(C, (m: DefinedMeth(C)))(n): bool =
    varnameNotSR(n) & n /= down(mtype(C,m))`parN
  VarNotSRpar(C, (m: DefinedMeth(C))): TYPE = (varNotSRpar(C,m))

  %----------------------------------------------------------------
  % Variable context - assignment of data types to variables/fields
  % (allowing it to be infinite saves some tccs)
  %----------------------------------------------------------------

% TBD could require that classT?( V`map(Self) ) if that helps any TCCs

  Vxt: TYPE = [# dom: set[Varname], map: [(dom)->dty] #]

  empty_Vxt: Vxt = (# dom:= emptyset, map:= (lambda (n:Varname): unitT) #)

  % fields of a class, as context
  fieldVxt(C: Classname): Vxt =
    (# `dom := { n | ftype(C,n) /= unitT },
```

```
      `map := lambda n: ftype(C,n) #)

  % context for arguments to method (self: C, parN: parT)
  methParVxt(C)(m: DefinedMeth(C)): Vxt =
    (# `dom := { n | n = Self OR n = down(mtype(C,m))`parN },
      `map := lambda n: IF   n = Self THEN classT(C)
                        ELSE               parType(C)(m) ENDIF #)

  % depth subclassing and equal domain for Vxt (only needed for self in meth pars)
  subclVxt(V1, V2: Vxt): bool =
    V1`dom = V2`dom &
    FORALL (n: (V1`dom)): V1`map(n) <= V2`map(n)

  subcl_methParVxt: LEMMA
    C <= D & definedMeth(D)(m) => subclVxt(methParVxt(C)(m),methParVxt(D)(m))

  %---------------------
  % simple consequences
  %---------------------

 meth_subcl: LEMMA
   D <= C & definedMeth(C)(m) => mtype(D,m) = mtype(C,m)

 super_methParVxt_dom: LEMMA
   FORALL (m: DefinedMeth(super(C))):
    methParVxt(C)(m)`dom = methParVxt(super(C))(m)`dom

 field_subcl: LEMMA
   D <= C & fields(C)(f) => fields(D)(f)

 field_subcl_V: LEMMA
   D <= C & fields(C)(f) => fieldVxt(D)`dom(f)

 field_subcl_Vf: LEMMA
   D <= C & fields(C)(f) => fieldVxt(D)`map(f) = ftype(C,f)

 resType_inh: LEMMA
   D <= C & definedMeth(C)(m)  =>  resType(C)(m) = resType(D)(m)

 parType_inh: LEMMA
   D <= C & definedMeth(C)(m)  =>  parType(C)(m) = parType(D)(m)

AUTO_REWRITE- leqClassname % avoid useless expansion of <= def in proofs

 END classtableSig
```

```
% $Revision: 1.2 $ $Date: 2004/03/08 05:33:38 $

typing: THEORY
 BEGIN
  IMPORTING
    classtableSig,
    lang[Classname, Varname, Methname]

 V:          VAR Vxt
 n, n1, n2: VAR Varname
 f:          VAR Varname
 T, T1, T2: VAR dty
 e, e1, e2: VAR exp
 S, S1, S2: VAR com
 C:          VAR Classname
 m:          VAR Methname

 %------------------------------------------------------------------
 % Ordinary typing for expressions, i.e., the relation V |- e:T
 % Note: subsumption is _not_ admissible due to "=" in test/cast/etc
 %------------------------------------------------------------------
 expOK(V,T)(e): RECURSIVE bool =
   CASES e OF
     vblV(n):   V`dom(n) & T = V`map(n) ,
     boolV(v):  T = boolT ,
     nullV:     classT?(T) ,
     eqtest(e1, T1, e2, T2): T = boolT
       & (T1=T2 & NOT classT?(T1) OR classT?(T1) & classT?(T2))
       & expOK(V,T1)(e1) & expOK(V,T2)(e2) ,
     fieldAccess(e1, T1, f): expOK(V,T1)(e1) & classT?(T1)
                         & fields(name(T1))(f)
                         & ftype(name(T1),f) = T ,
     typetest(e1, extype, checkedtype):
      classT?(extype) & T = boolT & checkedtype <= extype & expOK(V,extype)(e1),
     cast(e1, extype, casttotype):
       classT?(extype) & T = casttotype & T <= extype & expOK(V,extype)(e1)
  ENDCASES
  MEASURE e by <<

 %--------------------------------------------
 % Ordinary typing for commands, i.e., V |- S
 %--------------------------------------------
 comOK(V)(S): RECURSIVE bool =
   CASES S OF
    % n:= e
    assign(n, e, T): n /= Self & V`dom(n) & expOK(V,T)(e) & T <= V`map(n) ,
    % n:= new C
    new(n, C): n /= Self & V`dom(n) & classT(C) <= V`map(n) ,
    % e1.f:= e2
    fieldUpdate(e1, f, e2, T1, T2):
      expOK(V,T1)(e1) & expOK(V,T2)(e2) &
      classT?(T1) & fields(name(T1))(f)
                 & T2 <= ftype(name(T1),f) ,
```

```
    % n:= e1.m(e2)
    mcall(n, e1, m, e2, T1, T2):
      n /= Self & V`dom(n) & classT?(T1) & expOK(V,T1)(e1) & expOK(V,T2)(e2)
      & LET mty: lift[[# parN: Varname, parT: dty, resT: dty #]]
                 = mtype(name(T1),m) IN
         up?(mty) & T2 <= down(mty)`parT & down(mty)`resT <= V`map(n) ,
    ifelse(e, S1, S2): expOK(V,boolT)(e) & comOK(V)(S1) & comOK(V)(S2) ,
    seq(S1, S2): comOK(V)(S1) & comOK(V)(S2) ,
    skip: true
   ENDCASES
   MEASURE S by <<

 %---------------------------
 % Inversion of typing rules
 %---------------------------

 b: VAR bool

 var_in_V: LEMMA % note: would be subsumed by freeVarsOf?
  FORALL V,T, ( n | expOK(V,T)( vblV(n) ) ):
    V`dom(n)
 invert_vbl: LEMMA expOK(V,T)( vblV(n) ) =>  V`dom(n) & T = V`map(n)
 invert_bool: LEMMA expOK(V,T)( boolV(b) ) => T = boolT
 invert_null: LEMMA expOK(V,T)( nullV ) => classT?(T)
 invert_eqtest: LEMMA expOK(V,T)(eqtest(e1, T1, e2, T2)) =>
     T = boolT & expOK(V,T1)(e1) & expOK(V,T2)(e2)
     & (T1=T2 & NOT classT?(T1) OR classT?(T1) & classT?(T2))
 invert_fieldAccess: LEMMA expOK(V,T)(fieldAccess(e, T1, f)) =>
     expOK(V,T1)(e) & classT?(T1) & fields(name(T1))(f) & ftype(name(T1),f)=T
 invert_typetest: LEMMA expOK(V,T)(typetest(e, T1, T2)) =>
    classT?(T1) & T = boolT & T2 <= T1 & expOK(V,T1)(e)
 invert_cast: LEMMA expOK(V,T)(cast(e, T1, T2)) =>
    classT?(T1) & T2 <= T1 & expOK(V,T1)(e) & T2 = T
 invert_assign: LEMMA comOK(V)(assign(n, e, T)) =>
    n /= Self & V`dom(n) & expOK(V,T)(e) & T <= V`map(n)
 invert_new: LEMMA comOK(V)(new(n, C)) =>
    n /= Self & V`dom(n) & classT(C) <= V`map(n)
 invert_fieldUpdate: LEMMA comOK(V)(fieldUpdate(e1, f, e2, T1, T2)) =>
    expOK(V,T1)(e1) & expOK(V,T2)(e2)
    & classT?(T1) & fields(name(T1))(f) & T2 <= ftype(name(T1),f)
 invert_mcall: LEMMA comOK(V)(mcall(n, e1, m, e2, T1, T2)) =>
    n /= Self & V`dom(n) & classT?(T1) & expOK(V,T1)(e1) & expOK(V,T2)(e2)
    & LET mty: lift[[# parN: Varname, parT: dty, resT: dty #]]
                 = mtype(name(T1),m) IN
         up?(mty) & T2 <= down(mty)`parT & down(mty)`resT <= V`map(n)
 invert_ifelse: LEMMA comOK(V)(ifelse(e, S1, S2)) =>
    expOK(V,boolT)(e) & comOK(V)(S1) & comOK(V)(S2)
 invert_seq: LEMMA comOK(V)(seq(S1, S2)) =>
    comOK(V)(S1) & comOK(V)(S2)

 AUTO_REWRITE+  var_in_V % but not the inversions!

 END typing
```

```
wellformedCT: THEORY
 BEGIN

  IMPORTING typing

  x, n: VAR Varname

  T1, T2, T3: VAR dty

  V: VAR Vxt

  C, D: VAR Classname

  m: VAR Methname

  mbody_raw:
  [C: Classname, m: Methname →
     lift[[# localvar: Varname,
             localvarType: dty,
             body: com #]]]

  mbodyOK: AXIOM
    up?(mbody_raw(C, m)) & definedMeth(C)(m) ⇒
     varNotSRpar(C, m)(down(mbody_raw(C, m))‘localvar)

  mbody_range(C, ((m: DefinedMeth(C)))): TYPE =
  lift[[# localvar: VarNotSRpar(C, m),
          localvarType: dty,
          body: com #]]

  mbody(C: Classname, m: DefinedMeth(C)): mbody_range(C, m) =
      mbody_raw(C, m)

  declaredMeth(C)(m): bool =
      definedMeth(C)(m) & up?(mbody(C, m))

  DeclaredMeth(C): TYPE = (declaredMeth(C))

  every_meth_has_body: AXIOM
    definedMeth(C)(m) & ¬ declaredMeth(C)(m) ⇒
     C ≠ Object & definedMeth(super(C))(m)

  bodyVxtFor(C: Classname, m: DeclaredMeth(C)): Vxt =
      LET V = methParVxt(C)(m), n: VarNotSRpar(C, m) = down(mbody(C, m))‘localvar IN
        (# ‘dom := {x | V‘dom(x) ∨ x = Result ∨ x = n},
           ‘map
             := V‘map WITH [(n) |-> down(mbody(C, m))‘localvarType]
                  WITH [(Result) |-> resType(C)(m)] #)

  bodyTypable(C: Classname, m: DeclaredMeth(C)): bool =
      comOK(bodyVxtFor(C, m))(down(mbody(C, m))‘body)

  allBodiesTypable: AXIOM
```

```
∀ C, m: declaredMeth(C)(m) ⇒ bodyTypable(C, m)

declared_is_defined: LEMMA
  declaredMeth(C)(m) ⇒ definedMeth(C)(m)

END wellformedCT
```

```
value[Loc: TYPE+]: THEORY
 BEGIN

  value: DATATYPE
   BEGIN
    semLoc(valL: Loc): semLoc?
    semBool(valB: bool): semBool?
    semNil: semNil?
    semIt: semIt?
   END value

  l1, l2: VAR Loc

  b1, b2: VAR bool

  loc_inj: LEMMA semLoc(l1) = semLoc(l2) ⇒ l1 = l2

  semBool_inj: LEMMA semBool(b1) = semBool(b2) ⇒ b1 = b2

  valL_inj: LEMMA
    ∀ (v1, v2: (semLoc?)): valL(v1) = valL(v2) ⇒ v1 = v2

 END value
```

```
% $Revision: 1.6 $ $Date: 2004/03/21 23:44:43 $

semanticDomains: THEORY
%--------------------------------------------------------------------------
% values, stores, heaps, and meanings of expressions, commands, and methds
% The domains are straightforward but each needs one or more subsumption
% property.
%--------------------------------------------------------------------------
BEGIN
 IMPORTING classtableSig

 %------------------
 % memory locations
 %------------------

 Loc: TYPE+

 loctype: [Loc -> Classname]

 IMPORTING value[Loc]
 Value: TYPE = value[Loc]

 V:          VAR Vxt
 f, n, n1,n2:VAR Varname
 C, D:       VAR Classname
 T, T1:      VAR dty
 b:          VAR bool
 m:          VAR Methname
 v:          VAR Value

 locsBelow(C)(l: Loc): bool = loctype(l) <= C
 LocsBelow(C): TYPE = (locsBelow(C))

 %-------------
 % data values
 %-------------

 valOfType(T)(v: Value): bool =
  CASES T OF
   unitT:     semIt?(v) ,
   boolT:     semBool?(v) ,
   classT(C): semNil?(v) OR ( semLoc?(v) & locsBelow(C)(valL(v)) )
  ENDCASES
 ValOfType(T): TYPE = (valOfType(T))

 semBool: JUDGEMENT semBool(b) HAS_TYPE ValOfType(boolT)
 semIt: JUDGEMENT semIt HAS_TYPE ValOfType(unitT)
 locType: LEMMA
      FORALL (C:Classname, l: LocsBelow(C)): valOfType(classT(C))(semLoc(l))
 nilType: LEMMA valOfType(classT(C))(semNil)

 loc_val_below: LEMMA
   classT?(T) & valOfType(T)(v) & semLoc?(v) => loctype(valL(v)) <= name(T)
```

```
loc_val_type: LEMMA
  classT?(T) & valOfType(T)(v) & semLoc?(v) => valOfType(classT(loctype(valL(v))))(v)

val_subsumption: LEMMA
  C <= D & valOfType(classT( C ))(v) => valOfType(classT( D ))(v)

val_subsumptionT: LEMMA
  T <= T1 & valOfType(T)(v) => valOfType(T1)(v)

AUTO_REWRITE+ nilType, locType, loc_val_below, val_subsumption, val_subsumptionT

%------------------------------------------------------------------------
% Stores map names to values; closed/contained ones have nothing dangling
% Heaps map locations to suitable closed stores
%------------------------------------------------------------------------

Store(V): TYPE = [n: (V`dom) -> ValOfType(V`map(n))]

ObjState(C): TYPE = Store(fieldVxt(C))

preHeap: TYPE = [# dom: finite_set[Loc],
                   map: [ l:(dom) -> ObjState(loctype(l)) ] #]

closedStore(V)(h: preHeap)(r: Store(V)): bool =
  FORALL (n:(V`dom)):
    classT?(V`map(n)) & NOT semNil?(r(n)) => h`dom(valL(r(n)))

heap(h: preHeap): bool =
 FORALL ( l:(h`dom) ):
  LET V                      = fieldVxt(loctype(l)),
      r: ObjState(loctype(l)) = h`map(l)
   IN closedStore(V)(h)(r)
Heap: TYPE = (heap)

%---------------
% Global states
%---------------

closedStoreWithSelf(V: Vxt, h: Heap)(r: Store(V)): bool =
   closedStore(V)(h)(r) & V`dom(Self) & NOT semNil?(r(Self))

state(V)(h: Heap, r: Store(V)): bool = closedStoreWithSelf(V,h)(r)
State(V): TYPE = (state(V))

store_subsumpt: LEMMA
  FORALL (h: Heap), (V1, V2: Vxt), (r: Store(V1)):
    subclVxt(V1, V2) & closedStore(V1)(h)(r) => closedStore(V2)(h)(r)

state_subsumpt: LEMMA
  FORALL (V1, V2: Vxt), (s: (state(V1))):  subclVxt(V1,V2) => state(V2)(s)

methPar_subsumpt: LEMMA
```

```
  FORALL C, D, (m: DefinedMeth(D)), (s: State(methParVxt(C)(m))):
    C<=D  =>  state(methParVxt(D)(m))(s)

val_Vxt_subsumpt: LEMMA
  FORALL (V1, V2: Vxt), (n: (V1`dom)), (v: Value):
   subclVxt(V1,V2) & valOfType(V1`map(n))(v) => valOfType(V2`map(n))(v)

extend_heap_state: LEMMA
  FORALL (h1, h2: Heap), (V: Vxt), (r: Store(V)):
    subset?(h1`dom, h2`dom) & closedStore(V)(h1)(r) => closedStore(V)(h2)(r)

%-----------------------------------------------------------------
% expression meanings (any location value is contained in heap)
% TBD why contValOfType separate from closedStore
%-----------------------------------------------------------------

h: VAR Heap

preExpr(V,T): TYPE = [ State(V) -> lift[ValOfType(T)] ]

contValOfType(T, h)( v ): bool =
   valOfType(T)(v) &
   ( classT?(T) & NOT semNil?(v) => h`dom(valL(v)) )
CValOfType(T: dty, h: Heap): TYPE = (contValOfType(T,h))

contVal_subsumptionT: LEMMA
  T <= T1 & contValOfType(T,h)(v) => contValOfType(T1,h)(v)

liftContValOfType(T, h)( v: lift[Value] ): bool =
   up?(v) => contValOfType(T,h)(down(v))

bottom_closed: LEMMA liftContValOfType(T,h)(bottom)
nil_closed: LEMMA contValOfType(classT(C),h)(semNil)
bool_closed: LEMMA contValOfType(boolT,h)(semBool(b))

store_contained: LEMMA
  FORALL (V: Vxt, n: (V`dom), s: State(V)):
       contValOfType(V`map(n), s`1)(s`2(n))

AUTO_REWRITE+ store_contained, bottom_closed, nil_closed, bool_closed

% (Could define directly as
%   SemExpr(V,T): TYPE = [ s:State(V) -> lift[ CValOfType(T,s`1) ] ]
% but then it's hard to express subsumption.)

semExpr(V,T)(g: preExpr(V,T)): bool =
  FORALL (s:State(V)): liftContValOfType(T,s`1)(g(s))
SemExpr(V,T): TYPE = (semExpr(V,T))

sem_expr_subsumpt: LEMMA
  FORALL V,T,(g:SemExpr(V,T)):  T<=T1  =>  semExpr(V,T1)(g)

%------------------
```

```
% command meanings
% map states to lifted states,  where result heap domain extends initial heap
% (There's no use for it here where we aren't concerned with ref't, but
% prove lemma that definables don't modif Self, just as another sanity check.)
%------------------

preSemCommand(V): TYPE = [ State(V) -> lift[State(V)] ]

semCommand(V)(g: preSemCommand(V)): bool =
  FORALL (s: State(V), l:(s`1`dom)):  up?(g(s)) => down(g(s))`1`dom(l)
SemCommand(V): TYPE = (semCommand(V))

%-----------------
% method meanings map State(args) to bottom or a pair
% of type [h:Heap, CValOfType(resType(C)(m),h)] such that
% if the value is a location then it is in the domain of h.
%-----------------
% TBD rename methResult_pred to MethResult/methResult but in proofs too

methResult_pred(C,(m: DefinedMeth(C)))(h: Heap, v: Value): bool
  = contValOfType(resType(C)(m),h)(v)

methResult(C,(m: DefinedMeth(C))): TYPE = (methResult_pred(C,m))

methResult_subcl: LEMMA % (In fact equal due to invariant subtyping of methods.)
  FORALL C, D, (m: DefinedMeth(C)):
    D<= C  =>  methResult_pred(C,m) = methResult_pred(D,m)

% TBD Note: until increasing domain was added as condition in definition
% of SemMeth, it was possible to define semMeth like this:
%    semMeth(C: Classname, m: DefinedMeth(C))
%           (g: [State(methParVxt(C)(m)) -> lift[ methResult(C,m) ]]): bool = TRUE
% But I didn't see how to expand this definition in useful way for proofs.

semMeth(C: Classname, m: DefinedMeth(C))
       (g: [State(methParVxt(C)(m)) -> lift[[Heap,Value]]]): bool =
  FORALL (s:State(methParVxt(C)(m))):
    up?(g(s)) => ( methResult_pred(C,m)(down(g(s)))
                 & FORALL (l: (s`1`dom)): down(g(s))`1`dom(l) )

SemMeth(C: Classname, m: DefinedMeth(C)): TYPE = (semMeth(C,m))

semMeth_subsumpt: LEMMA
  FORALL C, D, (m: DefinedMeth(C)), (g: SemMeth(C,m)):
    D<=C => semMeth(D,m)(restrict[State(methParVxt(C)(m)),
                                  State(methParVxt(D)(m)), lift[methResult(C, m)]]
                                 (g))

%---------------------
% method environments
%---------------------

abortMeth(C: Classname, m: DefinedMeth(C)): SemMeth(C,m) =
```

```
  LET V = methParVxt(C)(m) IN  LAMBDA (s: State(V)):  bottom

 botMethEnv: [C: Classname -> [m: DefinedMeth(C) -> SemMeth(C,m)]] =
   LAMBDA C: LAMBDA (m: DefinedMeth(C)): abortMeth(C,m)

 MethEnv: NONEMPTY_TYPE = [C: Classname -> [m: DefinedMeth(C) -> SemMeth(C,m)]]

 %------------------------------------
 % initial values for data and object
 %------------------------------------

 default(T): ValOfType(T) =
   CASES T OF unitT: semIt, boolT: semBool(FALSE), classT(C): semNil ENDCASES

 initObState(C): ObjState(C) =
  LAMBDA (f: (fields(C))): default(ftype(C,f))

 default_closed: LEMMA contValOfType(T,h)( default(T) )
 AUTO_REWRITE+ default_closed

 init_closed: LEMMA closedStore(fieldVxt(C))(h)( initObState(C) )

 %---------------------
 % Boring consequences
 %---------------------

 val_subsumpt_x: LEMMA % TBD used any more?
   classT?(T) & classT?(T1) & valOfType(T)(v)
   & NOT semNil?(v) & loctype(valL(v))<=name(T1)
   =>  valOfType(T1)(v)

 field_type_simple: LEMMA
   FORALL (T: (classT?)), (f: (fields(name(T)))), (v: ValOfType(T)):
     NOT semNil?(v) => ftype(name(T),f) = ftype(loctype(valL(v)),f)
                     & fields(loctype(valL(v)))(f)

 field_type_V: LEMMA
    FORALL (T: (classT?)), (f: (fields(name(T)))), (v: ValOfType(T)):
     NOT semNil?(v) => fieldVxt(loctype(valL(v)))`dom(f)

 loc_val_in_heap: LEMMA
  FORALL (T: (classT?), V: Vxt, g: SemExpr(V, T), s: State(V)):
    up?(g(s)) & NOT down(g(s)) = semNil => s`1`dom(valL(down(g(s))))

 field_val_type_A: LEMMA
   FORALL V, (s: State(V)), (l: (s`1`dom)), (f: (fields(loctype(l)))):
     contValOfType(ftype(loctype(l),f),s`1)(s`1`map(l)(f))

% TBD methinks I could have used this in places where I used fieldUpdateS_TCCs
 field_val_type: LEMMA
   FORALL V, (s: State(V)), (T: (classT?)), (f: (fields(name(T)))),
          (lv: (liftContValOfType(T, s`1))):
     up?(lv) & NOT semNil?(down(lv))
```

```
    => contValOfType(ftype(name(T), f), s`1)(s`1`map(valL(down(lv)))(f))

 closedStore_inc_heap: LEMMA
   FORALL (r: Store(V)), (h1,h2: Heap):
     closedStore(V)(h1)(r) & subset?(h1`dom,h2`dom) => closedStore(V)(h2)(r)

 %-------------------------------------------------------
 % reformulate increasing heap domain in terms of subset
 %-------------------------------------------------------

 semCommand_result_subset: LEMMA
   FORALL (V: Vxt, g: SemCommand(V), s: State(V)):
     up?(g(s)) => subset?(s`1`dom, down(g(s))`1`dom)

 semMeth_result_subset: LEMMA
    FORALL (C: Classname, m: DefinedMeth(C), g: SemMeth(C,m), s: State(methParVxt(C)(m))):
      up?(g(s)) => subset?(s`1`dom, down(g(s))`1`dom)

END semanticDomains
```

```
% $Revision: 1.3 $ $Date: 2004/03/29 02:10:17 $

expSemantics: THEORY
 BEGIN
 IMPORTING semanticDomains, typing, lift_props
 CONVERSION- TotalFun_to_LPartFun

 V:          VAR Vxt
 T,T1,T2:    VAR dty
 n, n1, n2: VAR Varname
 e, e1, e2: VAR exp
 f:          VAR Varname
 C, D:       VAR Classname
 b:          VAR bool
 l:          VAR Loc

 %--------------------------------------------------------------------------
 % semantic operations for expression meanings, including errS for untypables
 %--------------------------------------------------------------------------

 errS(V,T)(s: State(V)): lift[CValOfType(T,s`1)] = bottom

 vblS(V)(n: (V`dom))(s: State(V)): lift[CValOfType(V`map(n), s`1 )]
   = up( s`2(n) )

 boolS(V)(b)(s: State(V)): lift[CValOfType(boolT, s`1)]
   = up( semBool(b) )

 nullS(V)(T:(classT?))(s: State(V)): lift[CValOfType(T, s`1)]
   = up( semNil )

 %(Defined for unrelated T1,T2 but typing rule is more restrictive.)
 eqtestS(V)(T1: dty, g1: SemExpr(V,T1), T2: dty, g2: SemExpr(V,T2))
          (s: State(V)): lift[CValOfType(boolT, s`1)] =
   LET v1: lift[Value] = g1(s),
       v2: lift[Value] = g2(s)
   IN IF bottom?(v1) OR bottom?(v2) THEN bottom
      ELSE up(semBool(down(v1) = down(v2))) ENDIF

 fieldAccessS(V)(T1: (classT?), g: SemExpr(V,T1), f: (fields(name(T1))))
             (s: State(V)): lift[CValOfType(ftype(name(T1),f), s`1 )] =
   LET v = g(s)
   IN  IF bottom?(v) OR down(v) = semNil THEN bottom
       ELSE LET l = valL(down(v)),
                ob: ObjState(loctype(l)) = s`1`map(l)
            IN up(ob(f))
       ENDIF

 typetestS(V)(T1: (classT?), g: SemExpr(V,T1), T2: Below(T1))
         (s: State(V)): lift[CValOfType( boolT, s`1 )] =
    LET v: lift[CValOfType(T1,s`1)] = g(s)
    IN IF bottom?(v) THEN bottom
       ELSIF down(v) = semNil THEN up(semBool(FALSE))
```

```
      ELSE  LET l = valL(down(v))
            IN up(semBool( loctype(l) <= name(T2) )) ENDIF

 castS(V)(T1: (classT?), g: SemExpr(V,T1), T2: Below(T1))
      (s: State(V)):  lift[CValOfType( T2, s`1 )] =
   LET v: lift[CValOfType(T1, s`1)] = g(s)
   IN IF bottom?(v) OR down(v) = semNil THEN bottom
      ELSIF loctype(valL(down(v))) <= name(T2) THEN v
      ELSE bottom ENDIF

 %------------------------------------------------------------
 % typing of semantic operations, for TCCs here and elsewhere
 %------------------------------------------------------------

 errS_type: LEMMA semExpr(V,T)( errS(V,T) )
 vblS_type: LEMMA V`dom(n) => semExpr(V,V`map(n))( vblS(V)(n) )
 boolS_type: LEMMA semExpr(V,boolT)( boolS(V)(b) )
 nullS_type: LEMMA classT?(T) => semExpr(V,T)( nullS(V)(T) )
 castS_type: LEMMA
   FORALL ( V: Vxt, T1: (classT?), T2: Below(T1), g: SemExpr(V,T1) ):
     semExpr(V,T2)( castS(V)(T1,g,T2) )
 eqtestS_type: LEMMA
   FORALL ( V:Vxt, T1:dty, T2:dty, g1: SemExpr(V,T1), g2: SemExpr(V,T2) ):
     semExpr(V, boolT)( eqtestS(V)(T1,g1,T2,g2) )
 fieldAccessS_type: LEMMA
   FORALL ( V: Vxt, T1: (classT?), g: SemExpr(V,T1), f: (fields(name(T1))) ):
     semExpr(V, ftype(name(T1),f))( fieldAccessS(V)(T1,g,f) )
 typetestS_type: LEMMA
   FORALL ( V: Vxt, T1: (classT?), g: SemExpr(V,T1), T2: Below(T1) ):
     semExpr(V, boolT)( typetestS(V)(T1,g,T2) )

 AUTO_REWRITE+ errS_type, vblS_type, boolS_type, nullS_type, castS_type,
   eqtestS_type, fieldAccessS_type, typetestS_type

 %------------------------------------------------------------
 % Interpretation of language
 % Defined even for untypable expressions, in order to use the
 % built-in measure and induction rule for the datatype 'exp'.
 %------------------------------------------------------------

%TBD redo tcc proofs esp simple cases where I used prop too soon and
% split unnecessarily; look at e.g. expSem_TCC13 proof for better.

 expSem(V,T)(e): RECURSIVE SemExpr(V,T) =
   IF NOT expOK(V,T)(e) THEN errS(V,T) ELSE
   CASES e OF
    vblV(n):  vblS(V)(n) ,
    boolV(b): boolS(V)(b) ,
    nullV:    nullS(V)(T) ,
    eqtest(e1, T1, e2, T2):
              eqtestS(V)(T1, expSem(V,T1)(e1), T2, expSem(V,T2)(e2) ) ,
    fieldAccess(e1, T1, f): fieldAccessS(V)(T1, expSem(V,T1)(e1), f) ,
    typetest(e1, T1, T2): typetestS(V)(T1, expSem(V,T1)(e1), T2) ,
```

```
    cast(e1, T1, T2): castS(V)(T1, expSem(V,T1)(e1), T2)
    ENDCASES ENDIF
    MEASURE e by <<

END expSemantics
```

```
comSemantics: THEORY
 BEGIN

  IMPORTING expSemantics

  CONVERSION- TotalFun_to_LPartFun


  V: VAR Vxt

  T, T1, T2: VAR dty

  f, n, n1, n2: VAR Varname

  C, D: VAR Classname

  m: VAR Methname

  e, e1, e2: VAR exp

  S, S1, S2: VAR com

  b: VAR bool

  l: VAR Loc

  me: VAR MethEnv

  h: VAR Heap

  v: VAR Value

  fresh: [Classname, Heap → Loc]

  freshness: AXIOM
    loctype(fresh(C, h)) = C & ¬ h`dom(fresh(C, h))

  updateVar(V)
           (n: {n | V`dom(n) & n ≠ Self}, s: State(V),
            v: CValOfType(V`map(n), s`1)):
  State(V) = s WITH [`2(n) := v]

  updateField(V)
             (s: State(V), l: (s`1`dom), f: (fields(loctype(l))),
              v: CValOfType(ftype(loctype(l), f), s`1)):
  State(V) =
      LET ob: ObjState(loctype(l)) = s`1`map(l),
          ob1: ObjState(loctype(l)) = ob WITH [(f) := v]
        IN s WITH [`1`map(l) := ob1]

  abortCom(V)(s: State(V)): lift[State(V)] = bottom

  argStore(C, ((m: DefinedMeth(C))), h: Heap,
```

```
        vSelf: {v | contValOfType(classT(C), h)(v) & ¬ semNil?(v)},
        vPar: CValOfType(parType(C)(m), h)):
 Store(methParVxt(C)(m)) =
     restrict[Varname, (methParVxt(C)(m)`dom), value[Loc]]
         (λ n: IF n = Self THEN vSelf ELSE vPar ENDIF)

 argStore_state: LEMMA
   ∀ (C: Classname, m: DefinedMeth(C), h: Heap,
      vSelf: {v | contValOfType(classT(C), h)(v) & ¬ semNil?(v)},
      vPar: CValOfType(parType(C)(m), h)):
     state(methParVxt(C)(m))
          (h, argStore(C, m, h, vSelf, vPar))

 updateVar_inc_heap: LEMMA
   ∀ (n: {n | V`dom(n) & n ≠ Self}, s: State(V), v: CValOfType(V`map(n), s`1),
      l: (s`1`dom)):
     updateVar(V)(n, s, v)`1`dom(l)

 updateVar_same_heap: LEMMA
   ∀ (n: {n | V`dom(n) & n ≠ Self}, s: State(V), v: CValOfType(V`map(n), s`1)):
     updateVar(V)(n, s, v)`1 = s`1

 updateField_same_dom: LEMMA
   ∀ (s: State(V), l: (s`1`dom), f: Fields(loctype(l)),
      v: CValOfType(ftype(loctype(l), f), s`1)):
     updateField(V)(s, l, f, v)`1`dom = s`1`dom

 assignS(V)(n: {n | V`dom(n) & n ≠ Self}, T: Below(V`map(n)), g: SemExpr(V, T))
        (s: State(V)):
 lift[State(V)] =
     IF up?(g(s))
       THEN up(updateVar(V)(n, s, down(g(s))))
     ELSE bottom
     ENDIF

 newS(V)(n: {n | V`dom(n) & n ≠ Self}, C: {C | below(V`map(n))(classT(C))})
     (s: State(V)):
 lift[State(V)] =
     LET l = fresh(C, s`1),
         h1: Heap =
            (# `dom := (s`1`dom ∪ {l}),
               `map := s`1`map WITH [(l) |-> initObState(C)] #)
       IN up(updateVar(V)(n, (h1, s`2), semLoc(l)))

 fieldUpdateS(V)
             (T1: (classT?), g1: SemExpr(V, T1), f: (fields(name(T1))),
              T2: Below(ftype(name(T1), f)), g2: SemExpr(V, T2))
             (s: State(V)):
 lift[State(V)] =
     LET v1 = g1(s), v2 = g2(s) IN
       IF bottom?(v1) ∨ semNil?(down(v1)) ∨ bottom?(v2)
         THEN bottom
       ELSE up(updateField(V)(s, valL(down(v1)), f, down(v2)))
```

```
    ENDIF

mcallS(V, me)
    (n: {n | V`dom(n) & n ≠ Self}, T1: (classT?), g1: SemExpr(V, T1),
     m: {m | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V`map(n)},
     g2: SemExpr(V, parType(name(T1))(m)))
    (s: State(V)):
lift[State(V)] =
  LET vTarg = g1(s), vArg = g2(s) IN
    IF bottom?(vTarg) ∨ semNil?(down(vTarg)) ∨ bottom?(vArg)
      THEN bottom
    ELSE LET dynClass = loctype(valL(down(vTarg))),
             args = argStore(dynClass, m, s`1, down(vTarg), down(vArg)),
             rslt = me(dynClass)(m)(s`1, args)
           IN
           IF bottom?(rslt)
             THEN bottom
           ELSE up(updateVar(V)(n, (down(rslt)`1, s`2), down(rslt)`2))
           ENDIF
    ENDIF

seqS(V)(g1, g2: SemCommand(V))(s: State(V)): lift[State(V)] =
  LET s1: lift[State(V)] = g1(s) IN
    IF bottom?(s1) THEN bottom ELSE g2(down(s1)) ENDIF

ifelseS(V)(g: SemExpr(V, boolT), g1, g2: SemCommand(V))(s: State(V)):
lift[State(V)] =
  LET v: lift[ValOfType(boolT)] = g(s) IN
    IF bottom?(v)
      THEN bottom
    ELSIF valB(down(v)) THEN g1(s)
    ELSE g2(s)
    ENDIF

skipS(V)(s: State(V)): lift[State(V)] = up(s)

abortCom_type: JUDGEMENT abortCom(V) HAS_TYPE
  SemCommand(V)

assignS_type: JUDGEMENT assignS(V)
                          (n: {n | V`dom(n) & n ≠ Self},
                           T: Below(V`map(n)), g: SemExpr(V, T))
  HAS_TYPE SemCommand(V)

newS_type: JUDGEMENT newS(V)
                       (n: {n | V`dom(n) & n ≠ Self},
                        C: {C | below(V`map(n))(classT(C))})
  HAS_TYPE SemCommand(V)

fieldS_type: JUDGEMENT fieldUpdateS(V)
                          (T1: (classT?), g1: SemExpr(V, T1),
                           f: (fields(name(T1))),
                           T2: Below(ftype(name(T1), f)),
```

```
                                    g2: SemExpr(V, T2))
    HAS_TYPE SemCommand(V)

  mcallS_type: JUDGEMENT mcallS(V, me)
                              (n: {n | V`dom(n) & n ≠ Self}, T1: (classT?),
                               g1: SemExpr(V, T1),
                               m:
                                 {m |
                                         definedMeth(name(T1))(m) &
                                          resType(name(T1))(m) ≤ V`map(n)},
                               g2: SemExpr(V, parType(name(T1))(m)))
    HAS_TYPE SemCommand(V)

  seqS_type: JUDGEMENT seqS(V)(g1, g2: SemCommand(V)) HAS_TYPE
    SemCommand(V)

  ifelseS_type: JUDGEMENT ifelseS(V)(g: SemExpr(V, boolT), g1, g2: SemCommand(V))
    HAS_TYPE SemCommand(V)

  skipS_type: JUDGEMENT skipS(V) HAS_TYPE SemCommand(V)

  mcallS_parType_type: LEMMA
    comOK(V)(mcall(n, e1, m, e2, T1, T2)) ⇒
     (∀ (s: State(V)):
        every[ValOfType(T2)]
            (restrict[value[Loc], ValOfType(T2), boolean]
                 (valOfType(parType(name(T1))(m))))
            (expSem(V, T2)(e2)(s)))
      ∧ semExpr(V, parType(name(T1))(m))(expSem(V, T2)(e2))

  comSem(V, me)(S): RECURSIVE SemCommand(V) =
    IF ¬ comOK(V)(S)
      THEN abortCom(V)
    ELSE CASES S
           OF assign(n, e, T): assignS(V)(n, T, expSem(V, T)(e)),
              new(n, C): newS(V)(n, C),
              fieldUpdate(e1, f, e2, T1, T2):
                fieldUpdateS(V)(T1, expSem(V, T1)(e1), f, T2, expSem(V, T2)(e2)),
              mcall(n, e1, m, e2, T1, T2):
                mcallS(V, me)(n, T1, expSem(V, T1)(e1), m, expSem(V, T2)(e2)),
              seq(S1, S2): seqS(V)(comSem(V, me)(S1), comSem(V, me)(S2)),
              ifelse(e, S1, S2):
                ifelseS(V)
                       (expSem(V, boolT)(e), comSem(V, me)(S1), comSem(V, me)(S2)),
              skip: skipS(V)
           ENDCASES
    ENDIF
     MEASURE S BY <<

 END comSemantics
```

```
orderDomains: THEORY
 BEGIN

  IMPORTING semanticDomains

  V: VAR Vxt

  C: VAR Classname

  T: VAR dty

  m: VAR Methname

  me1, me2: VAR MethEnv

  i, j: VAR nat

  CONVERSION- TotalFun_to_LPartFun


  leq(V)(s1, s2: lift[State(V)]): bool = up?(s1) ⇒ s1 = s2

  leq(V)(g1, g2: SemCommand(V)): bool =
      ∀ (s: State(V)): leq(V)(g1(s), g2(s))

  leq(C: Classname, m: DefinedMeth(C))(g1, g2: SemMeth(C, m)): bool =
      ∀ (s: State(methParVxt(C)(m))):
        up?(g1(s)) ⇒ g1(s) = g2(s)

  leq(me1, me2: MethEnv): bool =
      ∀ (C: Classname, m: DefinedMeth(C)):
        leq(C, m)(me1(C)(m), me2(C)(m))

  ascending_menvs(mes: [nat → MethEnv]): bool =
      ∀ i, (j: upfrom(i)): leq(mes(i), mes(j))

  AscendingMenvs: TYPE = (ascending_menvs)

  lub(mes: AscendingMenvs): MethEnv =
      λ C:
        λ (m: DefinedMeth(C)):
          λ (s: State(methParVxt(C)(m))):
            IF ∃ j: up?(mes(j)(C)(m)(s))
              THEN LET i = min(λ j: up?(mes(j)(C)(m)(s))) IN mes(i)(C)(m)(s)
            ELSE bottom
            ENDIF

  bot_leq_state: LEMMA
    ∀ (s: lift[State(V)]): leq(V)(bottom, s)

  abortMeth_bot: LEMMA
    ∀ C, (m: DefinedMeth(C)), (g: SemMeth(C, m)):
      leq(C, m)(abortMeth(C, m), g)
```

```
botMethEnv_bot: LEMMA leq(botMethEnv, me1)

asc_menvs_step: LEMMA
  ∀ (mes: [nat → MethEnv]):
    (∀ i: leq(mes(i), mes(i + 1))) ⇒
     ascending_menvs(mes)

leq_com_reflexive: LEMMA
  ∀ (g: SemCommand(V)): leq(V)(g, g)

leq_meth_reflexive: LEMMA
  ∀ (m: DefinedMeth(C)), (g: SemMeth(C, m)):
    leq(C, m)(g, g)

characterize_menv: LEMMA
  leq(me1, me2) ⇔
   (∀ (C: Classname, m: DefinedMeth(C), s: State(methParVxt(C)(m))):
      up?(me1(C)(m)(s)) ⇒ me1(C)(m)(s) = me2(C)(m)(s))

characterize_asc_menvs: LEMMA
  ∀ (j: nat, i: upfrom(j), mes: AscendingMenvs, C: Classname, m: DefinedMeth(C),
     s: State(methParVxt(C)(m))):
    up?(mes(j)(C)(m)(s)) ⇒
     mes(j)(C)(m)(s) = mes(i)(C)(m)(s)

characterize_lub_bot: LEMMA
  ∀ (mes: AscendingMenvs, C: Classname, m: DefinedMeth(C),
     s: State(methParVxt(C)(m))):
    bottom?(lub(mes)(C)(m)(s)) ⇒
     ¬ (∃ j: up?(mes(j)(C)(m)(s)))

characterize_menv_lub: LEMMA
  ∀ (mes: AscendingMenvs), (m: DefinedMeth(C)), (s: State(methParVxt(C)(m))):
    ∃ j:
      ∀ (i: upfrom(j)):
        lub(mes)(C)(m)(s) = mes(i)(C)(m)(s)

END orderDomains
```

```
comSemanticsProps: THEORY
 BEGIN

  IMPORTING comSemantics, orderDomains

  seqS_mono: LEMMA
    ∀ (V: Vxt), (g1, g2, g1?, g2?: SemCommand(V)):
      leq(V)(g1, g1?) & leq(V)(g2, g2?) ⇒
       leq(V)(seqS(V)(g1, g2), seqS(V)(g1?, g2?))

  ifelseS_mono: LEMMA
    ∀ (V: Vxt), (g1, g2, g1?, g2?: SemCommand(V)), (g: SemExpr(V, boolT)):
      leq(V)(g1, g1?) & leq(V)(g2, g2?) ⇒
       leq(V)
          (ifelseS(V)(g, g1, g2), ifelseS(V)(g, g1?, g2?))

  me, me1, me2: VAR MethEnv

  V: VAR Vxt

  S: VAR com

  n: VAR Varname

  m: VAR Methname

  mcall_mono: LEMMA
    ∀ (V: Vxt, n: {n | V`dom(n) & n ≠ Self}, T1: (classT?), g1: SemExpr(V, T1),
       m: {m | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V`map(n)},
       g2: SemExpr(V, parType(name(T1))(m))):
      leq(me1, me2) ⇒
       leq(V)
          (mcallS(V, me1)(n, T1, g1, m, g2),
           mcallS(V, me2)(n, T1, g1, m, g2))

  comSem_mono: LEMMA
    ∀ V, me1, me2, S:
      leq(me1, me2) ⇒
       leq(V)(comSem(V, me1)(S), comSem(V, me2)(S))

 END comSemanticsProps
```

```
semCT: THEORY
 BEGIN

  IMPORTING wellformedCT, comSemanticsProps

  Value: TYPE = semanticDomains.Value

  CONVERSION- TotalFun_to_LPartFun


  V: VAR Vxt

  n, local: VAR Varname

  C, D: VAR Classname

  T: VAR dty

  m: VAR Methname

  S: VAR com

  h: VAR Heap

  me, me1, me2: VAR MethEnv

  initMbodyStore(C, ((m: DeclaredMeth(C))), s: State(methParVxt(C)(m))):
  (closedStoreWithSelf(bodyVxtFor(C, m), s`1)) =
      LET local: VarNotSRpar(C, m) = down(mbody(C, m))`localvar IN
        s`2
          WITH [(local) |-> default(down(mbody(C, m))`localvarType),
                (Result) |-> default(resType(C)(m))]

  initMbodyState(C, ((m: DeclaredMeth(C))), ((s: State(methParVxt(C)(m))))):
  State(bodyVxtFor(C, m)) = (s`1, initMbodyStore(C, m, s))

  extractResult(C, ((m: DeclaredMeth(C))), ((s: lift[State(bodyVxtFor(C, m))]))):
  lift[methResult(C, m)] =
      IF s = bottom
        THEN bottom
      ELSE up(down(s)`1, down(s)`2(Result))
      ENDIF

  semComToMeth(C, ((m: DeclaredMeth(C))), ((g: SemCommand(bodyVxtFor(C, m)))))
               (s: State(methParVxt(C)(m))):
  lift[methResult(C, m)] =
      extractResult(C, m, g(initMbodyState(C, m, s)))

  semComToMeth_type: JUDGEMENT semComToMeth(C: Classname, m: DeclaredMeth(C),
                                              g: SemCommand(bodyVxtFor(C, m)))
      HAS_TYPE SemMeth(C, m)

  mbodySem(C, me)(m: DeclaredMeth(C)): SemMeth(C, m) =
```

```
    LET S = down(mbody(C, m))`body IN
      semComToMeth(C, m, comSem(bodyVxtFor(C, m), me)(S))

  i, j: VAR nat

  inherit_defined: LEMMA
    ∀ C, (m: DefinedMeth(super(C))), (g: SemMeth(super(C), m)):
      semMeth(C, m)
             (restrict[State(methParVxt(super(C))(m)), State(methParVxt(C)(m)),
                       lift[methResult(super(C), m)]]
                   (g))

  approxMethEnv(j)(C): RECURSIVE [m: DefinedMeth(C) → SemMeth(C, m)] =
    λ (m: DefinedMeth(C)):
      IF j = 0
        THEN abortMeth(C, m)
      ELSE IF declaredMeth(C)(m)
             THEN mbodySem(C, approxMethEnv(j − 1))(m)
           ELSE restrict[State(methParVxt(super(C))(m)), State(methParVxt(C)(m)),
                         lift[methResult(super(C), m)]]
                  (approxMethEnv(j)(super(C))(m))
           ENDIF
      ENDIF
     MEASURE lex2(j, cdepth(C))
     BY restrict[[ordstruct, ordstruct], [ordinal, ordinal], boolean]
          (<)

  semComToMeth_mono: LEMMA
    ∀ (m: DeclaredMeth(C)):
      LET V = bodyVxtFor(C, m) IN
        ∀ (g, g?: SemCommand(V)):
          leq(V)(g, g?) ⇒
           leq(C, m)
              (semComToMeth(C, m, g),
               semComToMeth(C, m, g?))

  mbodySem_mono: LEMMA
    ∀ me1, me2, (m: DeclaredMeth(C)):
      leq(me1, me2) ⇒
       leq(C, m)(mbodySem(C, me1)(m), mbodySem(C, me2)(m))

  ascending_approx_step: LEMMA
    leq(approxMethEnv(j), approxMethEnv(j + 1))

  ascending_approx: JUDGEMENT approxMethEnv HAS_TYPE
      AscendingMenvs

  semCT: MethEnv = lub(approxMethEnv)

END semCT
```

```
safe: THEORY
 BEGIN

  IMPORTING wellformedCT, lattice

  CONVERSION- TotalFun_to_LPartFun


  flowPolicy:
  [Classname, Methname →
     [# self: Level, par: Level, res: Level, hp: Level #]]

  flowPolicyFld: [Classname → [Varname → Level]]

  localLevel: [Classname, Methname, Varname → Level]

  C, D: VAR Classname

  f: VAR Varname

  m: VAR Methname

  inherit_field_lev: AXIOM
    C ≤ D & ftype(D, f) ≠ unitT ⇒
     flowPolicyFld(C)(f) = flowPolicyFld(D)(f)

  inherit_meth_lev: AXIOM
    C ≤ D & up?(mtype(D, m)) ⇒
     flowPolicy(C, m) = flowPolicy(D, m)

  V: VAR Vxt

  n, n1, n2: VAR Varname

  T, T1, T2: VAR dty

  e, e1, e2: VAR exp

  S, S1, S2: VAR com

  L, L1, L2, L3, L4: VAR Level

  LL, LL1, LL2, LL3: VAR lift[Level]

  L1p, L2p: VAR [Level, Level]

  LL1p, LL2p: VAR lift[[Level, Level]]

  Levels(V): TYPE = [(V`dom) → Level]

  expSafe(V: Vxt, lev: Levels(V), T: dty)(e: exp): RECURSIVE lift[Level] =
    IF ¬ expOK(V, T)(e)
      THEN bottom
```

```
  ELSE CASES e
         OF vblV(n): up(lev(n)),
            boolV(v): up(botL),
            nullV: up(botL),
            eqtest(e1, T1, e2, T2):
              s_lub(expSafe(V, lev, T1)(e1), expSafe(V, lev, T2)(e2)),
            fieldAccess(e1, T1, f):
              s_lub(expSafe(V, lev, T1)(e1), up(flowPolicyFld(name(T1))(f))),
            typetest(e1, T1, T2): expSafe(V, lev, T1)(e1),
            cast(e1, T1, T2): expSafe(V, lev, T1)(e1)
         ENDCASES
  ENDIF
   MEASURE e BY <<

comSafe(V: Vxt, levs: Levels(V), S: com): RECURSIVE lift[[Level, Level]] =
  IF ¬ comOK(V)(S)
    THEN bottom
  ELSE CASES S
         OF assign(n, e, T):
              LET L = expSafe(V, levs, T)(e) IN
                IF up?(L) & down(L) ≤ levs(n)
                  THEN up(levs(n), topL)
                ELSE bottom
                ENDIF,
            fieldUpdate(e1, f, e2, T1, T2):
              LET L1 = expSafe(V, levs, T1)(e1),
                  L2 = expSafe(V, levs, T2)(e2),
                  L = flowPolicyFld(name(T1))(f)
                IN
                IF up?(L1) & up?(L2) & lub(down(L1), down(L2)) ≤ L
                  THEN up(topL, L)
                ELSE bottom
                ENDIF,
            new(n, C): up(levs(n), topL),
            mcall(n, e1, m, e2, T1, T2):
              LET LL1 = expSafe(V, levs, T1)(e1),
                  LL2 = expSafe(V, levs, T2)(e2),
                  policy = flowPolicy(name(T1), m)
                IN
                IF up?(LL1) &
                     up?(LL2) &
                      down(LL1) ≤ policy`self &
                       down(LL2) ≤ policy`par &
                        policy`res ≤ levs(n) &
                         policy`self ≤ policy`res &
                          policy`self ≤ policy`hp & policy`self ≤ levs(n)
                  THEN up(levs(n), policy`hp)
                ELSE bottom
                ENDIF,
            ifelse(e, S1, S2):
              LET L = expSafe(V, levs, boolT)(e),
                  L1p = comSafe(V, levs, S1),
                  L2p = comSafe(V, levs, S2)
```

```
                    IN
                    IF L = bottom ∨ L1p = bottom ∨ L2p = bottom
                      THEN bottom
                    ELSE LET L1 = glb(down(L1p)‘1, down(L2p)‘1),
                             L2 = glb(down(L1p)‘2, down(L2p)‘2)
                           IN
                           IF down(L) ≤ glb(L1, L2) THEN up(L1, L2) ELSE bottom ENDIF
                    ENDIF,
                seq(S1, S2):
                  LET L1p = comSafe(V, levs, S1), L2p = comSafe(V, levs, S2) IN
                    IF L1p = bottom ∨ L2p = bottom
                      THEN bottom
                    ELSE up(glb(down(L1p)‘1, down(L2p)‘1),
                            glb(down(L1p)‘2, down(L2p)‘2))
                    ENDIF,
                skip: up(topL, topL)
            ENDCASES
    ENDIF
     MEASURE S BY <<

  invert_safe_fieldAccess: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(expSafe(V, lev, T)(fieldAccess(e1, T1, f))) ⇒
       up?(expSafe(V, lev, T1)(e1))

  invert_safe_eqtest: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(expSafe(V, lev, T)(eqtest(e1, T1, e2, T2))) ⇒
       up?(expSafe(V, lev, T1)(e1)) &
        up?(expSafe(V, lev, T2)(e2))

  invert_safe_typetest: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(expSafe(V, lev, T)(typetest(e1, T1, T2))) ⇒
       up?(expSafe(V, lev, T1)(e1))

  invert_safe_cast: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(expSafe(V, lev, T)(cast(e1, T1, T2))) ⇒
       up?(expSafe(V, lev, T1)(e1))

  invert_safe_assign: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(comSafe(V, lev, assign(n, e, T))) ⇒
       up?(expSafe(V, lev, T)(e)) &
        down(expSafe(V, lev, T)(e)) ≤ lev(n)

  invert_safe_fieldUpdate: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(comSafe(V, lev, fieldUpdate(e1, f, e2, T1, T2))) ⇒
       up?(expSafe(V, lev, T1)(e1)) &
        up?(expSafe(V, lev, T2)(e2)) &
         lub(down(expSafe(V, lev, T1)(e1)), down(expSafe(V, lev, T2)(e2))) ≤
```

```
      flowPolicyFld(name(T1))(f)

  invert_safe_mcall: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(comSafe(V, lev, mcall(n, e1, m, e2, T1, T2))) ⇒
       LET LL1 = expSafe(V, lev, T1)(e1),
           LL2 = expSafe(V, lev, T2)(e2),
           policy = flowPolicy(name(T1), m)
         IN
         up?(LL1) &
          up?(LL2) &
           down(LL1) ≤ policy‘self &
            down(LL2) ≤ policy‘par &
             policy‘res ≤ lev(n) &
              policy‘self ≤ policy‘hp &
               policy‘self ≤ policy‘res &
                policy‘self ≤ lev(n)

  invert_safe_ifelse: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(comSafe(V, lev, ifelse(e, S1, S2))) ⇒
       up?(expSafe(V, lev, boolT)(e)) &
        up?(comSafe(V, lev, S1)) &
         up?(comSafe(V, lev, S2)) &
          down(expSafe(V, lev, boolT)(e)) ≤
           glb(glb(down(comSafe(V, lev, S1))‘1, down(comSafe(V, lev, S2))‘1),
               glb(down(comSafe(V, lev, S1))‘2,
                   down(comSafe(V, lev, S2))‘2))

  invert_safe_seq: LEMMA
    ∀ (V: Vxt, lev: Levels(V)):
      up?(comSafe(V, lev, seq(S1, S2))) ⇒
       up?(comSafe(V, lev, S1)) & up?(comSafe(V, lev, S2))

  methParLevels(C: Classname, m: DefinedMeth(C)): Levels(methParVxt(C)(m)) =
      LET V = methParVxt(C)(m),
          policy: [# self: Level, par: Level, res: Level, hp: Level #] =
             flowPolicy(C, m)
        IN
        λ (n: (V‘dom)):
          IF n = Self
            THEN policy‘self
          ELSE policy‘par
          ENDIF

  mbodyLevels(C: Classname, m: DeclaredMeth(C)): Levels(bodyVxtFor(C, m)) =
      LET V = bodyVxtFor(C, m),
          policy: [# self: Level, par: Level, res: Level, hp: Level #] =
             flowPolicy(C, m)
        IN
        λ (n: (V‘dom)):
          IF n = Self
            THEN policy‘self
```

```
        ELSIF n = down(mtype(C, m))`parN THEN policy`par
        ELSIF n = Result THEN policy`res
        ELSE localLevel(C, m, n)
        ENDIF

 mSafe(C)(m: DeclaredMeth(C)): bool =
    LET V = bodyVxtFor(C, m),
        levs: Levels(V) = mbodyLevels(C, m),
        L1p = comSafe(V, levs, down(mbody(C, m))`body)
      IN up?(L1p) & flowPolicy(C, m)`hp ≤ down(L1p)`2

 safe_CT: bool = ∀ C, (m: DeclaredMeth(C)): mSafe(C)(m)

 safe_OK_exp: LEMMA
   ∀ V, (lev: Levels(V)):
     up?(expSafe(V, lev, T)(e)) ⇒ expOK(V, T)(e)

 safe_OK_com: LEMMA
   ∀ V, (lev: Levels(V)):
     up?(comSafe(V, lev, S)) ⇒ comOK(V)(S)

END safe
```

```
indistinguishable: THEORY
 BEGIN

  IMPORTING lattice, semanticDomains

  CONVERSION- TotalFun_to_LPartFun


  l, l' , l_0, l_0' : VAR Loc

  C, D: VAR Classname

  V: VAR Vxt

  methPolicy: TYPE =
  [Classname, Methname →
     [# self: Level, par: Level, res: Level, hp: Level #]]

  fieldPolicy: TYPE = [Classname → [Varname → Level]]

  Levels(V): TYPE = [(V`dom) → Level]

  fpMeth: VAR methPolicy

  fpField: VAR fieldPolicy

  T, T_1: VAR dty

  L, L_1, L_2, L_3, L_4: VAR Level

  n, f: VAR Varname

  typedRel(r: pred[[Loc, Loc]]): bool =
      ∀ l, l' : r(l, l' ) ⇒ loctype(l) = loctype(l' )

  bijRel(r: pred[[Loc, Loc]]): bool =
      ∀ l, l' , l_0, l_0' :
        r(l, l' ) & r(l_0, l_0' ) ⇒ (l = l_0 ⇔ l'  = l_0' )

  typBij(ρ: pred[[Loc, Loc]]): bool =
      typedRel(ρ) & bijRel(ρ)

  TypBij: TYPE = (typBij)

  ρ, τ: VAR TypBij

  h, h' , h_0, h_1, h_2, h_3: VAR Heap

  ls, ls2: VAR set[Loc]

  left_dom(ρ)(l): bool = ∃ (l' ): ρ(l, l' )

  rt_dom(ρ)(l): bool = ∃ (l' ): ρ(l' , l)
```

```
idRel(ls: set[Loc])(l, l′ ): bool = ls(l) & l = l′

idRel_typBij: JUDGEMENT idRel(ls: set[Loc]) HAS_TYPE
    TypBij

extendBij(ρ, l, l′ )(l0, l_0′ ): bool =
    ρ(l0, l_0′ ) ∨ l0 = l & l_0′  = l′

extendBij_typ: JUDGEMENT extendBij(ρ, ((l: Loc | ¬ (left_dom(ρ)(l)))),
                                     ((l′ : Loc
                                           | loctype(l′ ) = loctype(l)
                                             &
                                             ¬ (rt_dom(ρ)(l′ )))))
    HAS_TYPE TypBij

extendBij_subset: LEMMA (ρ ⊆ extendBij(ρ, l, l′ ))

left_dom_id: LEMMA left_dom(idRel(ls)) = ls

rt_dom_id: LEMMA rt_dom(idRel(ls)) = ls

dom_rng_typBij: LEMMA
  (left_dom(ρ) ⊆ h‘dom) & (rt_dom(ρ) ⊆ h′ ‘dom) & ρ(l, l′ ) ⇒
   h‘dom(l) & h′ ‘dom(l′ )

typ_bij_comp: JUDGEMENT O(ρ, τ) HAS_TYPE TypBij

dom_bij_comp: LEMMA (left_dom(ρ ∘ τ) ⊆ left_dom(ρ))

rng_bij_comp: LEMMA (rt_dom(ρ ∘ τ) ⊆ rt_dom(τ))

typBij_type: LEMMA ρ(l, l′ ) ⇒ loctype(l) = loctype(l′ )

subset_pointwise: LEMMA (ρ ⊆ τ) & ρ(l, l′ ) ⇒ τ(l, l′ )

indis(ρ, T)(v, v′ : ValOfType(T)): bool =
    CASES T
      OF unitT: TRUE,
         boolT: v = v′ ,
         classT(C):
           (semNil?(v) & semNil?(v′ )) ∨
            (semLoc?(v) & semLoc?(v′ ) & ρ(valL(v), valL(v′ )))
      ENDCASES

indis(ρ, T)(v, v′ : lift[ValOfType(T)]): bool =
    (bottom?(v) & bottom?(v′ )) ∨
     (up?(v) & up?(v′ ) & indis(ρ, T)(down(v), down(v′ )))

indis_default: LEMMA indis(ρ, T)(default(T), default(T))

indis(L, ρ, V, ((lev: Levels(V))))(s, s′ : Store(V)): bool =
    ∀ ((n: Varname | V‘dom(n) & lev(n) ≤ L)):
```

```
    indis(ρ, V‘map(n))(s(n), s′ (n))

indis(L, ρ, C, fpField)(s, s′ : ObjState(C)): bool =
  indis(L, ρ, fieldVxt(C),
      restrict[Varname, (fieldVxt(C)‘dom), Level](fpField(C)))
     (s, s′ )

indis(L, ρ, fpField)(h, h′ ): bool =
  (left_dom(ρ) ⊆ h‘dom) &
   (rt_dom(ρ) ⊆ h′ ‘dom) &
    (∀ l, l′ :
      ρ(l, l′ ) ⇒
       indis(L, ρ, fieldVxt(loctype(l)),
          restrict[Varname, (fieldVxt(loctype(l))‘dom), Level]
            (fpField(loctype(l))))
          (h‘map(l), h′ ‘map(l′ )))

indis(L, ρ, V, fpField, ((lev: Levels(V))))(s, s′ : State(V)): bool =
  indis(L, ρ, fpField)(s‘1, s′ ‘1) &
   indis(L, ρ, V, lev)(s‘2, s′ ‘2)

indis(L, ρ, fpField, L2, C, ((m: DefinedMeth(C))))(r, r′ : methResult(C, m)): bool =
  indis(L, ρ, fpField)(r‘1, r′ ‘1) &
   (L2 ≤ L ⇒ indis(ρ, resType(C)(m))(r‘2, r′ ‘2))

readConf(L, V, T, fpField)(levs: Levels(V))(g: SemExpr(V, T)): bool =
  ∀ ρ, (s, s′ : State(V)):
    indis(L, ρ, V, fpField, levs)(s, s′ ) ⇒
     indis(ρ, T)(g(s), g(s′ ))

Lstore, Lheap: VAR Level

writeConf(Lstore, Lheap, V, fpField, ((lev: Levels(V))))(g: SemCommand(V)): bool =
  ∀ L1, L2, (s: State(V)):
    up?(g(s)) ⇒
     LET ι = idRel(s‘1‘dom), s0 = down(g(s)) IN
      (¬ (Lheap ≤ L1) ⇒ indis(L1, ι, fpField)(s‘1, s0‘1)) &
       (¬ (Lstore ≤ L2) ⇒
         indis(L2, ι, V, lev)(s‘2, s0‘2))

writeConfMeth(C, fpMeth, fpField)(m: DefinedMeth(C))(g: SemMeth(C, m)): bool =
  LET V = methParVxt(C)(m) IN
    ∀ L, (s: State(V)):
      up?(g(s)) & ¬ (fpMeth(C, m)‘hp ≤ L) ⇒
       LET ι = idRel(s‘1‘dom), h0 = down(g(s))‘1 IN
        indis(L, ι, fpField)(s‘1, h0)

me: VAR MethEnv

writeConfMenv2(fpMeth, fpField)(me)(C): bool =
  ∀ (m: DefinedMeth(C)):
    writeConfMeth(C, fpMeth, fpField)(m)(me(C)(m))
```

```
writeConfMenv(fpMeth, fpField)(me): bool =
    ∀ C: writeConfMenv2(fpMeth, fpField)(me)(C)

indis_trans_store: LEMMA
  ∀ (V: Vxt, s1, s2, s3: Store(V), lev: Levels(V)):
    indis(L, ρ, V, lev)(s1, s2) & indis(L, τ, V, lev)(s2, s3) ⇒
     indis(L, (ρ ∘ τ), V, lev)(s1, s3)

indis_trans_heap: LEMMA
  indis(L, ρ, fpField)(h1, h2) & indis(L, τ, fpField)(h2, h3) ⇒
   indis(L, (ρ ∘ τ), fpField)(h1, h3)

idRel_comp_left: LEMMA idRel(left_dom(ρ)) ∘ ρ = ρ

idRel_comp_rt: LEMMA
  (rt_dom(ρ) ⊆ ls) ⇒ ρ ∘ idRel(ls) = ρ

idRel_comp_subset: LEMMA
  (ls ⊆ ls2) ⇒
   idRel(ls) ∘ idRel(ls2) = idRel(ls)

idRel_comp_left_subset: LEMMA
  (left_dom(ρ) ⊆ ls) ⇒ idRel(ls) ∘ ρ = ρ

idRel_comp_rt_subset: LEMMA
  (rt_dom(ρ) ⊆ ls) ⇒ ρ ∘ idRel(ls) = ρ

indis_sym_id_store: LEMMA
  ∀ (V: Vxt, ((s1), (s2), (s3: Store(V))), lev: Levels(V), ls: set[Loc]):
    indis(L, idRel(ls), V, lev)(s1, s2) ⇒
     indis(L, idRel(ls), V, lev)(s2, s1)

indis_sym_id_heap: LEMMA
  indis(L, idRel(h1‘dom), fpField)(h1, h2) ⇒
   indis(L, idRel(h1‘dom), fpField)(h2, h1)

indis_sym_id_state: LEMMA
  ∀ L, (ls: set[Loc]), V, fpField, (lev: Levels(V)), (s, s′ : State(V)):
    indis(L, idRel(ls), V, fpField, lev)(s, s′ ) ⇒
     indis(L, idRel(ls), V, fpField, lev)(s′ , s)

indis_id_refl_store: LEMMA
  ∀ L, V, (lev: Levels(V)), h, (s: (closedStore(V)(h))):
    indis(L, idRel(h‘dom), V, lev)(s, s)

indis_id_refl_heap: LEMMA
  ∀ L, fpField, h: indis(L, idRel(h‘dom), fpField)(h, h)

indis_id_refl_state: LEMMA
  ∀ L, V, fpField, (lev: Levels(V)), (s: State(V)):
    indis(L, idRel(s‘1‘dom), V, fpField, lev)(s, s)

indis_trans_store_ifelse: LEMMA
```

```
  ∀ (V: Vxt, s_0, s_1, s_2, s_3: Store(V), lev: Levels(V), ls0, ls1: set[Loc]):
    (left_dom(ρ) ⊆ ls0) &
     (rt_dom(ρ) ⊆ ls1) &
      indis(L, idRel(ls0), V, lev)(s_0, s_1) &
       indis(L, ρ, V, lev)(s_1, s_2) & indis(L, idRel(ls1), V, lev)(s_2, s_3)
     ⇒ indis(L, ρ, V, lev)(s_0, s_3)

indis_trans_heap_ifelse: LEMMA
  ∀ (ls0, ls1: set[Loc]):
    (left_dom(ρ) ⊆ ls0) &
     (rt_dom(ρ) ⊆ ls1) &
      indis(L, idRel(ls0), fpField)(h_0, h_1) &
       indis(L, ρ, fpField)(h_1, h_2) & indis(L, idRel(ls1), fpField)(h_2, h_3)
     ⇒ indis(L, ρ, fpField)(h_0, h_3)

indis_mono_val: LEMMA
  ∀ ρ, T, T_1, (v, v' : ValOfType(T)):
    T ≤ T_1 & indis(ρ, T_1)(v, v' ) ⇒ indis(ρ, T)(v, v' )

indis_antimono_val: LEMMA
  ∀ ρ, T, T_1, (v, v' : ValOfType(T_1)):
    T_1 ≤ T & indis(ρ, T_1)(v, v' ) ⇒ indis(ρ, T)(v, v' )

indis_antimono_store: LEMMA
  ∀ (V: Vxt, ((s_1), (s_2: Store(V))), lev: Levels(V)):
    L_1 ≤ L_2 & indis(L_2, ρ, V, lev)(s_1, s_2) ⇒
     indis(L_1, ρ, V, lev)(s_1, s_2)

indis_antimono_heap: LEMMA
  L_1 ≤ L_2 & indis(L_2, ρ, fpField)(h_1, h_2) ⇒
   indis(L_1, ρ, fpField)(h_1, h_2)

indis_antimono_state: LEMMA
  ∀ (V: Vxt, ((s_1), (s_2: State(V))), lev: Levels(V)):
    L_1 ≤ L_2 & indis(L_2, ρ, V, fpField, lev)(s_1, s_2) ⇒
     indis(L_1, ρ, V, fpField, lev)(s_1, s_2)

indis_mono_bij_store: LEMMA
  ∀ (V: Vxt, ((s_1), (s_2: Store(V))), lev: Levels(V)):
    indis(L, ρ, V, lev)(s_1, s_2) & (ρ ⊆ τ) ⇒
     indis(L, τ, V, lev)(s_1, s_2)

indis_mono_result: LEMMA
  ∀ (m: DefinedMeth(D)), (r, r' : methResult(D, m)):
    C ≤ D & indis(L, ρ, fpField, L_2, D, m)(r, r' ) ⇒
     indis(L, ρ, fpField, L_2, C, m)(r, r' )

indis_antimono_rel_store: LEMMA
  ∀ (V: Vxt, ((s_1), (s_2: Store(V))), lev: Levels(V)):
    (ρ ⊆ τ) & indis(L, ρ, V, lev)(s_1, s_2) ⇒
     indis(L, τ, V, lev)(s_1, s_2)

indis_res_yields_indis_call: LEMMA
```

```
  ∀ L,
    V,
    ρ,
    (lev: Levels(V)),
    L₂, C, (m: DefinedMeth(C)), (s, s′ : State(V)), (r, r′ : methResult(C, m)):
    indis(L, ρ, V, fpField, lev)(s, s′ ) &
     indis(L, τ, fpField, L₂, C, m)(r, r′ ) &
      (ρ ⊆ τ) & (s‘1‘dom ⊆ r‘1‘dom) & (s′ ‘1‘dom ⊆ r′ ‘1‘dom)
     ⇒
     indis(L, τ, V, fpField, lev)((r‘1, s‘2), (r′ ‘1, s′ ‘2))

indis_extend_heap_pres_store: LEMMA
  ∀ L, V, ρ, τ, (lev: Levels(V)), (s, s′ : State(V)), (h, h′ : Heap):
    indis(L, ρ, V, fpField, lev)(s, s′ ) &
     indis(L, τ, fpField)(h, h′ ) &
      (ρ ⊆ τ) & (s‘1‘dom ⊆ h‘dom) & (s′ ‘1‘dom ⊆ h′ ‘dom)
     ⇒ indis(L, τ, V, fpField, lev)((h, s‘2), (h′ , s′ ‘2))

read_conf_mono: LEMMA
  ∀ V, T, (g: SemExpr(V, T)), (lev: Levels(V)):
    L₁ ≤ L₂ & readConf(L₁, V, T, fpField)(lev)(g) ⇒
     readConf(L₂, V, T, fpField)(lev)(g)

write_conf_antimono: LEMMA
  ∀ (V: Vxt, g: SemCommand(V), lev: Levels(V)):
    L₁ ≤ L₂ & L₃ ≤ L₄ & writeConf(L₂, L₄, V, fpField, lev)(g) ⇒
     writeConf(L₁, L₃, V, fpField, lev)(g)

read_conf_mono_type: LEMMA
  ∀ V, L, T, T₁, (g: SemExpr(V, T)), (lev: Levels(V)):
    T ≤ T₁ & readConf(L, V, T, fpField)(lev)(g) ⇒
     readConf(L, V, T₁, fpField)(lev)(g)

END indistinguishable
```

```
confinement: THEORY
 BEGIN

  IMPORTING indistinguishable, semCT, safe

  CONVERSION- TotalFun_to_LPartFun


  L, L1, L2, L3, L4: VAR Level

  ρ, τ: VAR TypBij

  h0, h1, h2, h3: VAR Heap

  l: VAR Loc

  ι: VAR TypBij

  fpMeth: VAR methPolicy

  fpField: VAR fieldPolicy

  T, T1, T2: VAR dty

  b: VAR bool

  C, D: VAR Classname

  m: VAR Methname

  V: VAR Vxt

  n, f: VAR Varname

  Levels(V): TYPE = indistinguishable.Levels(V)

  invarSubcl(fp: fieldPolicy): bool =
      ∀ C, D, f:
         C ≤ D & ftype(D, f) ≠ unitT ⇒
          fp(C)(f) = fp(D)(f)

  invarSubcl(fp: methPolicy): bool =
      ∀ C, D, m:
         C ≤ D & up?(mtype(D, m)) ⇒ fp(C, m) = fp(D, m)

  invar_subcl_field: LEMMA invarSubcl(flowPolicyFld)

  invar_subcl_meth: LEMMA invarSubcl(flowPolicy)

  errS_read_conf: LEMMA
    ∀ V, (levs: Levels(V)):
      readConf(L, V, T, fpField)(levs)(errS(V, T))
```

```
boolS_read_conf: LEMMA
  ∀ V, (levs: Levels(V)):
    readConf(L, V, boolT, fpField)(levs)(boolS(V)(b))

nullS_read_conf: LEMMA
  ∀ (V: Vxt, levs: Levels(V), T: (classT?)):
    readConf(L, V, T, fpField)(levs)(nullS(V)(T))

vblS_read_conf: LEMMA
  ∀ (V: Vxt, levs: Levels(V)):
    V‘dom(n) & V‘map(n) = T & levs(n) ≤ L ⇒
     readConf(L, V, T, fpField)(levs)(vblS(V)(n))

eqtest_read_conf_lem: LEMMA
  ∀ T1, T2, (v, v' : ValOfType(T1)), (v2, v_2' : ValOfType(T2)):
    (T1 = T2 & ¬ classT?(T1) ∨ classT?(T1) & classT?(T2)) &
     indis(ρ, T1)(v, v' ) & indis(ρ, T2)(v2, v_2' )
     ⇒
     indis(ρ, boolT)
          (up(semBool(v = v2)), up(semBool(v'  = v_2' )))

eqtestS_read_conf: LEMMA
  ∀ V, T1, T2, (levs: Levels(V)), (g1: SemExpr(V, T1)), (g2: SemExpr(V, T2)):
    (T1 = T2 & ¬ classT?(T1) ∨ classT?(T1) & classT?(T2)) &
     readConf(L1, V, T1, fpField)(levs)(g1) &
      readConf(L2, V, T2, fpField)(levs)(g2)
     ⇒
     readConf(lub(L1, L2), V, boolT, fpField)(levs)
              (eqtestS(V)(T1, g1, T2, g2))

fieldAccessS_read_conf: LEMMA
  ∀ (V: Vxt, T1: (classT?), g: SemExpr(V, T1), f: (fields(name(T1))),
     levs: Levels(V)):
    readConf(L, V, T1, fpField)(levs)(g) & invarSubcl(fpField) ⇒
     readConf(lub(L, fpField(name(T1))(f)), V, ftype(name(T1), f), fpField)(levs)
              (fieldAccessS(V)(T1, g, f))

typetestS_read_conf: LEMMA
  ∀ (V: Vxt, T1: (classT?), g: SemExpr(V, T1), levs: Levels(V), T2: Below(T1)):
    readConf(L, V, T1, fpField)(levs)(g) ⇒
     readConf(L, V, boolT, fpField)(levs)
              (typetestS(V)(T1, g, T2))

castS_read_conf: LEMMA
  ∀ (V: Vxt, T1: (classT?), g: SemExpr(V, T1), levs: Levels(V), T2: Below(T1)):
    readConf(L, V, T1, fpField)(levs)(g) ⇒
     readConf(L, V, T2, fpField)(levs)
              (castS(V)(T1, g, T2))

me: VAR MethEnv

writeConfMenv(me): bool =
    writeConfMenv(flowPolicy, flowPolicyFld)(me)
```

```
WriteConfMenv: TYPE = (writeConfMenv)

updateVar_write_conf: LEMMA
  ∀ V,
    L,
    fpField,
    (lev: Levels(V)),
    ((n: (V`dom) | n ≠ Self)), (s: State(V)), (v: CValOfType(V`map(n), s`1)):
    LET ι = idRel(s`1`dom), s0 = updateVar(V)(n, s, v) IN
      indis(topL, ι, fpField)(s`1, s0`1) ∧
       (¬ lev(n) ≤ L ⇒ indis(L, ι, V, lev)(s`2, s0`2))

extend_heap_write_conf: LEMMA
  ∀ (V: Vxt, lev: Levels(V), s: State(V), ((n: (V`dom) | n ≠ Self)),
     C: {C | below(V`map(n))(classT(C))}):
    LET l = fresh(C, s`1),
        h1 =
           (# `dom := (s`1`dom ∪ {l}),
              `map := s`1`map WITH [(l) −−> initObState(C)] #),
        ι = idRel(s`1`dom)
      IN indis(topL, ι, V, fpField, lev)(s, (h1, s`2))

updateField_write_conf: LEMMA
  ∀ V,
    fpField,
    (lev: Levels(V)),
    (s: State(V)),
    (l: (s`1`dom)),
    (f: (fields(loctype(l)))), (v: CValOfType(ftype(loctype(l), f), s`1)):
    LET ι = idRel(s`1`dom), s0 = updateField(V)(s, l, f, v) IN
      indis(topL, ι, V, lev)(s`2, s0`2) ∧
       (¬ fpField(loctype(l))(f) ≤ L ⇒
         indis(L, ι, fpField)(s`1, s0`1))

mcallS_heap_effect_write_conf: LEMMA
  ∀ (V: Vxt, lev: Levels(V), ((n: (V`dom) | n ≠ Self)), T1: (classT?),
     g1: SemExpr(V, T1),
     ((m: Methname
           | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V`map(n))),
     g2: SemExpr(V, parType(name(T1))(m)), me: WriteConfMenv,
     ((L: Level | ¬ flowPolicy(name(T1), m)`hp ≤ L)), s: State(V)):
    LET vTarg = g1(s), vArg = g2(s) IN
      up?(vTarg) & ¬ semNil?(down(vTarg)) & up?(vArg) ⇒
       LET dynClass = loctype(valL(down(vTarg))),
           args = argStore(dynClass, m, s`1, down(vTarg), down(vArg)),
           rslt = me(dynClass)(m)(s`1, args)
         IN
         up?(rslt) ⇒
          indis(L, idRel(s`1`dom), flowPolicyFld)
               (s`1, down(rslt)`1)

abortCom_write_conf: LEMMA
```

```
  ∀ (V: Vxt, lev: Levels(V)):
    writeConf(L1, L2, V, fpField, lev)(abortCom(V))

seqS_write_conf: LEMMA
  ∀ (V: Vxt, ((g1), (g2: SemCommand(V))), lev: Levels(V)):
    writeConf(L1, L2, V, fpField, lev)(g1) &
     writeConf(L3, L4, V, fpField, lev)(g2)
     ⇒
     writeConf(glb(L1, L3), glb(L2, L4), V, fpField, lev)
              (seqS(V)(g1, g2))

ifelseS_write_conf: LEMMA
  ∀ (V: Vxt, ((g1), (g2: SemCommand(V))), lev: Levels(V), g: SemExpr(V, boolT)):
    writeConf(L1, L2, V, fpField, lev)(g1) &
     writeConf(L3, L4, V, fpField, lev)(g2)
     ⇒
     writeConf(glb(L1, L3), glb(L2, L4), V, fpField, lev)
              (ifelseS(V)(g, g1, g2))

assignS_write_conf: LEMMA
  ∀ (V: Vxt, lev: Levels(V), ((n: (V`dom) | n ≠ Self)), T: Below(V`map(n)),
     g: SemExpr(V, T)):
    writeConf(lev(n), topL, V, fpField, lev)
             (assignS(V)(n, T, g))

newS_write_conf: LEMMA
  ∀ (V: Vxt, lev: Levels(V), ((n: (V`dom) | n ≠ Self)),
     C: {C | below(V`map(n))(classT(C))}):
    writeConf(lev(n), topL, V, fpField, lev)
             (newS(V)(n, C))

fieldUpdateS_write_conf: LEMMA
  ∀ (V: Vxt, T1: (classT?), g1: SemExpr(V, T1), f: (fields(name(T1))),
     T2: Below(ftype(name(T1), f)), g2: SemExpr(V, T2), lev: Levels(V)):
    invarSubcl(fpField) ⇒
     writeConf(topL, fpField(name(T1))(f), V, fpField, lev)
              (fieldUpdateS(V)(T1, g1, f, T2, g2))

mcallS_write_conf: LEMMA
  ∀ (V: Vxt, lev: Levels(V), ((n: (V`dom) | n ≠ Self)), T1: (classT?),
     g1: SemExpr(V, T1),
     ((m: Methname
           | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V`map(n))),
     g2: SemExpr(V, parType(name(T1))(m)), me: WriteConfMenv):
    writeConf(lev(n), flowPolicy(name(T1), m)`hp, V, flowPolicyFld, lev)
             (mcallS(V, me)(n, T1, g1, m, g2))

skipS_write_conf: LEMMA
  ∀ (lev: Levels(V)):
    writeConf(L1, L2, V, fpField, lev)(skipS(V))

writeConfMeth_mono: LEMMA
  ∀ C, D, (m: DefinedMeth(D)), (g: SemMeth(D, m)):
```

```
     C ≤ D & writeConfMeth(D, flowPolicy, flowPolicyFld)(m)(g) ⇒
      writeConfMeth(C, flowPolicy, flowPolicyFld)(m)
                  (restrict(g))

  write_conf_abortMeth: LEMMA
    ∀ C, fpMeth, fpField, (m: DefinedMeth(C)):
      writeConfMeth(C, fpMeth, fpField)(m)(abortMeth(C, m))

  safe_exp_read_conf: LEMMA
    ∀ (V: Vxt, levs: Levels(V), T: dty, e: exp):
      LET LL = expSafe(V, levs, T)(e) IN
        up?(LL) ⇒
         readConf(down(LL), V, T, flowPolicyFld)(levs)
                 (expSem(V, T)(e))

  j: VAR nat

  LLp: VAR lift[[Level, Level]]

  safe_com_write_conf: LEMMA
    ∀ (V: Vxt, lev: Levels(V), S: com, me: WriteConfMenv):
      LET LLp = comSafe(V, lev, S) IN
        up?(LLp) ⇒
         writeConf(down(LLp)`1, down(LLp)`2, V, flowPolicyFld, lev)
                  (comSem(V, me)(S))

  safe_mbody_write_conf: LEMMA
    ∀ C, ((m: Methname | declaredMeth(C)(m) & mSafe(C)(m))), (me: WriteConfMenv):
      writeConfMeth(C, flowPolicy, flowPolicyFld)(m)
                   (mbodySem(C, me)(m))

  write_conf_admiss: LEMMA
    ∀ (app: AscendingMenvs):
      (∀ j: writeConfMenv(app(j))) ⇒
       writeConfMenv(lub(app))

  write_conf_approx: LEMMA
    safe_CT ⇒ (∀ j: writeConfMenv(approxMethEnv(j)))

  safe_menv_write_conf: LEMMA
    safe_CT ⇒ writeConfMenv(semCT)

END confinement
```

```
safeProps: THEORY
 BEGIN

  IMPORTING confinement, int_props

  CONVERSION- TotalFun_to_LPartFun


  C, D: VAR Classname

  f: VAR Varname

  m: VAR Methname

  L, L1, L2, L3, L4, L_rslt: VAR Level

  LL: VAR lift[Level]

  LLp: VAR lift[[Level, Level]]

  V: VAR Vxt

  n, n1, n2: VAR Varname

  T, T1, T2: VAR dty

  e, e1, e2: VAR exp

  S, S1, S2: VAR com

  b: VAR bool

  me: VAR MethEnv

  i, j: VAR nat

  ρ, τ: VAR TypBij

  l, l' , l0, l_0' : VAR Loc

  Levels(V): TYPE = indistinguishable.Levels(V)

  nonint(L, V, ((lev: Levels(V))))(g: SemCommand(V)): bool =
      ∀ ρ, (s, s' : State(V)):
        indis(L, ρ, V, flowPolicyFld, lev)(s, s' ) & up?(g(s)) & up?(g(s' )) ⇒
         (∃ τ:
            (ρ ⊆ τ) &
             indis(L, τ, V, flowPolicyFld, lev)
                  (down(g(s)), down(g(s' ))))

  nonint(L, C, ((m: DefinedMeth(C))))(g: SemMeth(C, m)): bool =
      LET L_rslt = flowPolicy(C, m)`res IN
        ∀ ρ, (s, s' : State(methParVxt(C)(m))):
```

```
        indis(L, ρ, methParVxt(C)(m), flowPolicyFld, methParLevels(C, m))(s, s' ) &
         up?(g(s)) & up?(g(s' ))
         ⇒
         (∃ τ:
            (ρ ⊆ τ) &
             indis(L, τ, flowPolicyFld, L_rslt, C, m)
                  (down(g(s)), down(g(s' ))))

nonintMenv2(me, C): bool =
    ∀ L, (m: DefinedMeth(C)): nonint(L, C, m)(me(C)(m))

nonintMenv(me): bool = ∀ C: nonintMenv2(me, C)

indis_mono_methParVxt_A: LEMMA
  ∀ C, (V1, V2: Vxt), (s, s' : State(V1)), (lev: Levels(V1)):
    subclVxt(V1, V2) & indis(L, ρ, V2, flowPolicyFld, lev)(s, s' ) ⇒
     indis(L, ρ, V1, flowPolicyFld, lev)(s, s' )

indis_antimono_methParVxt_A: LEMMA
  ∀ C, (V1, V2: Vxt), (s, s' : State(V1)), (lev: Levels(V1)):
    subclVxt(V1, V2) & indis(L, ρ, V1, flowPolicyFld, lev)(s, s' ) ⇒
     indis(L, ρ, V2, flowPolicyFld, lev)(s, s' )

indis_mono_methParVxt_B: LEMMA
  ∀ C, (m: DefinedMeth(super(C))):
    methParLevels(C, m) = methParLevels(super(C), m)

indis_mono_methParVxt: LEMMA
  ∀ C, (m: DefinedMeth(super(C))), (s, s' : State(methParVxt(C)(m))):
    indis(L, ρ, methParVxt(super(C))(m), flowPolicyFld, methParLevels(super(C), m))
         (s, s' )
     ⇒
     indis(L, ρ, methParVxt(C)(m), flowPolicyFld, methParLevels(C, m))
          (s, s' )

indis_antimono_methParVxt: LEMMA
  ∀ C, (m: DefinedMeth(super(C))), (s, s' : State(methParVxt(C)(m))):
    indis(L, ρ, methParVxt(C)(m), flowPolicyFld, methParLevels(C, m))(s, s' ) ⇒
     indis(L, ρ, methParVxt(super(C))(m), flowPolicyFld,
           methParLevels(super(C), m))
          (s, s' )

nonint_super: LEMMA
  ∀ L, C, (m: DefinedMeth(super(C))), (g: SemMeth(super(C), m)):
    nonint(L, super(C), m)(g) ⇒
     nonint(L, C, m)(restrict(g))

nonint_updateVar: LEMMA
  ∀ L,
    V,
    ρ,
    (lev: Levels(V)),
    ((n: (V`dom) | n ≠ Self)),
```

```
    (s, s' : State(V)),
    (v: CValOfType(V`map(n), s`1)), (v' : CValOfType(V`map(n), s' `1)):
    (lev(n) ≤ L ⇒ indis(ρ, V`map(n))(v, v' )) &
     indis(L, ρ, V, flowPolicyFld, lev)(s, s' )
     ⇒
     indis(L, ρ, V, flowPolicyFld, lev)
          (updateVar(V)(n, s, v), updateVar(V)(n, s' , v' ))

nonint_newS_ext_typ: LEMMA
  ∀ V, (s, s' : State(V)), (lev: Levels(V)):
    indis(L, ρ, V, flowPolicyFld, lev)(s, s' ) ⇒
     typBij(extendBij(ρ, fresh(C, s`1), fresh(C, s' `1)))

extend_heap_nonint: LEMMA
  ∀ (V: Vxt, L: Level, lev: Levels(V), ((s), (s' : State(V))),
     ((n: (V`dom) | n ≠ Self)), ((C: Classname | below(V`map(n))(classT(C))))):
    indis(L, ρ, V, flowPolicyFld, lev)(s, s' ) ⇒
     LET l = fresh(C, s`1),
         l'  = fresh(C, s' `1),
         h =
            (# `dom := (s`1`dom ∪ {l}),
               `map := s`1`map WITH [(l) —-> initObState(C)] #),
         h'  =
            (# `dom := (s' `1`dom ∪ {l' }),
               `map := s' `1`map WITH [(l' ) —-> initObState(C)] #),
         τ = extendBij(ρ, l, l' )
       IN indis(L, τ, flowPolicyFld)(h, h' )

nonint_updateField_not_visible: LEMMA
  ∀ L,
    V,
    (lev: Levels(V)),
    (T: (classT?)),
    (f: (fields(name(T)))),
    (s, s' : State(V)),
    ((l: (s`1`dom) | loctype(l) ≤ name(T))),
    ((l' : (s' `1`dom) | loctype(l' ) ≤ name(T))),
    (v: (contValOfType(ftype(name(T), f), s`1))),
    (v' : (contValOfType(ftype(name(T), f), s' `1))):
    indis(L, ρ, V, flowPolicyFld, lev)(s, s' ) & ¬ flowPolicyFld(name(T))(f) ≤ L ⇒
     indis(L, ρ, V, flowPolicyFld, lev)
          (updateField(V)(s, l, f, v),
           updateField(V)(s' , l' , f, v' ))

initMbodyState_indis: LEMMA
  ∀ L, ρ, C, (m: DeclaredMeth(C)), (s, s' : State(methParVxt(C)(m))):
    indis(L, ρ, methParVxt(C)(m), flowPolicyFld, methParLevels(C, m))(s, s' ) ⇒
     indis(L, ρ, bodyVxtFor(C, m), flowPolicyFld, mbodyLevels(C, m))
          (initMbodyState(C, m, s),
           initMbodyState(C, m, s' ))

extractResult_indis: LEMMA
  ∀ C, (m: DeclaredMeth(C)), (s, s' : lift[State(bodyVxtFor(C, m))]):
```

```
    up?(s) &
     up?(s′ ) &
      indis(L, ρ, bodyVxtFor(C, m), flowPolicyFld, mbodyLevels(C, m))
           (down(s), down(s′ ))
     ⇒
     indis(L, ρ, flowPolicyFld, flowPolicy(C, m)‘res, C, m)
          (down(extractResult(C, m, s)),
           down(extractResult(C, m, s′ )))

nonint_abort: LEMMA
  ∀ L, C, (m: DefinedMeth(C)):
    nonint(L, C, m)(abortMeth(C, m))

nonint_skip: LEMMA
  ∀ L, V, (lev: Levels(V)): nonint(L, V, lev)(skipS(V))

nonint_assignS: LEMMA
  ∀ L,
    V,
    ((n: (V‘dom) | n ≠ Self)),
    (T: Below(V‘map(n))), (g: SemExpr(V, T)), (lev: Levels(V)):
    readConf(lev(n), V, T, flowPolicyFld)(lev)(g) ⇒
     nonint(L, V, lev)(assignS(V)(n, T, g))

nonint_newS: LEMMA
  ∀ L,
    V,
    ((n: (V‘dom) | n ≠ Self)),
    (lev: Levels(V)), ((C: Classname | below(V‘map(n))(classT(C)))):
    nonint(L, V, lev)(newS(V)(n, C))

nonint_seqS: LEMMA
  ∀ L, V, (g1, g2: SemCommand(V)), (lev: Levels(V)):
    nonint(L, V, lev)(g1) & nonint(L, V, lev)(g2) ⇒
     nonint(L, V, lev)(seqS(V)(g1, g2))

nonint_fieldUpdateS: LEMMA
  ∀ L,
    V,
    (lev: Levels(V)),
    (T1: (classT?)),
    (g1: SemExpr(V, T1)),
    (f: (fields(name(T1)))),
    (T2: Below(ftype(name(T1), f))), (g2: SemExpr(V, T2)):
    readConf(flowPolicyFld(name(T1))(f), V, T1, flowPolicyFld)(lev)(g1) &
     readConf(flowPolicyFld(name(T1))(f), V, T2, flowPolicyFld)(lev)(g2)
     ⇒
     nonint(L, V, lev)(fieldUpdateS(V)(T1, g1, f, T2, g2))

nonint_mcallS_args: LEMMA
  ∀ L,
    V,
    ρ,
```

```
    (lev: Levels(V)),
    me,
    ((n: (V`dom) | n ≠ Self)),
    (T1: (classT?)),
    (g1: SemExpr(V, T1)),
    ((m: Methname | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V`map(n))),
    (g2: SemExpr(V, parType(name(T1))(m))), (s, s' : State(V)):
    LET vTarg = g1(s), vTarg? = g1(s' ), vArg = g2(s), vArg? = g2(s' ) IN
      up?(vTarg) &
       ¬ semNil?(down(vTarg)) &
        up?(vArg) &
         up?(vTarg?) &
          ¬ semNil?(down(vTarg?)) &
           up?(vArg?) &
            indis(L, ρ, V, flowPolicyFld, lev)(s, s' ) &
             flowPolicy(name(T1), m)`self ≤ L &
              readConf(flowPolicy(name(T1), m)`self, V, T1, flowPolicyFld)(lev)
                      (g1)
               &
               readConf(flowPolicy(name(T1), m)`par, V, parType(name(T1))(m),
                        flowPolicyFld)
                       (lev)(g2)
       ⇒
       LET dynClass = loctype(valL(down(vTarg))),
           dynClass? = loctype(valL(down(vTarg?))),
           args = argStore(dynClass, m, s`1, down(vTarg), down(vArg)),
           args? = argStore(dynClass?, m, s' `1, down(vTarg?), down(vArg?)),
           Varg = methParVxt(dynClass)(m)
         IN
         indis(L, ρ, Varg, flowPolicyFld, methParLevels(dynClass, m))
              ((s`1, args), (s' `1, args?))

nonint_mcallS_res_type: LEMMA
  ∀ (Lobs: Level, V: Vxt, lev: Levels(V), ((n: (V`dom) | n ≠ Self)),
     T1: (classT?), g1: SemExpr(V, T1),
     m: {m | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V`map(n)},
     g2: SemExpr(V, parType(name(T1))(m)), s: State(V),
     ((me: MethEnv | nonintMenv(me) & writeConfMenv(me)))):
    up?(g1(s)) &
     up?(g2(s)) &
      ¬ semNil?(down(g1(s))) &
       up?(me(loctype(valL(down(g1(s)))))(m)
             (s`1,
              argStore(loctype(valL(down(g1(s)))), m, s`1, down(g1(s)),
                       down(g2(s)))))
     ⇒
     valOfType(resType(loctype(valL(down(g1(s)))))(m))
              (down(me(loctype(valL(down(g1(s)))))(m)
                      (s`1,
                       argStore(loctype(valL(down(g1(s)))), m, s`1, down(g1(s)),
                                down(g2(s)))))`2)

nonint_mcallS_vis: LEMMA
```

```
∀ (Lobs: Level, V: Vxt, lev: Levels(V), ((n: (V‘dom) | n ≠ Self)),
   T1: (classT?), g1: SemExpr(V, T1),
   m: {m | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V‘map(n)},
   g2: SemExpr(V, parType(name(T1))(m)), ((s), (s' : State(V))),
   ((me: MethEnv | nonintMenv(me) & writeConfMenv(me)))):
  flowPolicy(name(T1), m)‘self ≤ Lobs &
   flowPolicy(name(T1), m)‘res ≤ lev(n) &
    flowPolicy(name(T1), m)‘self ≤ lev(n) &
     flowPolicy(name(T1), m)‘self ≤ flowPolicy(name(T1), m)‘hp &
      readConf(flowPolicy(name(T1), m)‘self, V, T1, flowPolicyFld)(lev)(g1) &
       readConf(flowPolicy(name(T1), m)‘par, V, parType(name(T1))(m),
                flowPolicyFld)
               (lev)(g2)
        &
        indis(Lobs, ρ, V, flowPolicyFld, lev)(s, s' ) &
         up?(mcallS(V, me)(n, T1, g1, m, g2)(s)) &
          up?(mcallS(V, me)(n, T1, g1, m, g2)(s' ))
   ⊃
   bottom?(g1(s)) ∨
    semNil?(down(g1(s))) ∨
     bottom?(g2(s)) ∨
      bottom?(me(loctype(valL(down(g1(s)))))(m)
                (s‘1,
                 argStore(loctype(valL(down(g1(s)))), m, s‘1, down(g1(s)),
                          down(g2(s)))))
       ∨
       bottom?(g1(s' )) ∨
        semNil?(down(g1(s' ))) ∨
         bottom?(g2(s' )) ∨
          bottom?(me(loctype(valL(down(g1(s' )))))(m)
                    (s' ‘1,
                     argStore(loctype(valL(down(g1(s' )))), m, s' ‘1, down(g1(s' )),
                              down(g2(s' )))))
           ∨
           (∃ τ:
              (ρ ⊆ τ) &
               indis(Lobs, τ, V, flowPolicyFld, lev)
                    (updateVar(V)
                              (n,
                               (down(me(loctype(valL(down(g1(s)))))(m)
                                       (s‘1,
                                        argStore(loctype(valL(down(g1(s)))),
                                                 m,
                                                 s‘1,
                                                 down(g1(s)),
                                                 down(g2(s)))))‘1,
                                s‘2),
                               down(me(loctype(valL(down(g1(s)))))(m)
                                      (s‘1,
                                       argStore(loctype(valL(down(g1(s)))),
                                                m,
                                                s‘1,
                                                down(g1(s)),
```

```
                                                    down(g2(s)))))‘2),
                                  updateVar(V)
                                         (n,
                                          (down(me(loctype(valL(down(g1(s′ )))))(m)
                                                 (s′ ‘1,
                                                  argStore(loctype(valL(down(g1(s′ )))),
                                                          m,
                                                          s′ ‘1,
                                                          down(g1(s′ )),
                                                          down(g2(s′ )))))‘1,
                                           s′ ‘2),
                                          down(me(loctype(valL(down(g1(s′ )))))(m)
                                                (s′ ‘1,
                                                 argStore(loctype(valL(down(g1(s′ )))),
                                                         m,
                                                         s′ ‘1,
                                                         down(g1(s′ )),
                                                         down
                                                         (g2
                                                          (s′ )))))‘2)))

nonint_mcallS_invis: LEMMA
  ∀ (Lobs: Level, V: Vxt, lev: Levels(V), ((n: (V‘dom) | n ≠ Self)),
     T1: (classT?), g1: SemExpr(V, T1),
     m: {m | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V‘map(n)},
     g2: SemExpr(V, parType(name(T1))(m)), ((s), (s′ : State(V))),
     ((me: MethEnv | nonintMenv(me) & writeConfMenv(me)))):
    flowPolicy(name(T1), m)‘res ≤ lev(n) &
     flowPolicy(name(T1), m)‘self ≤ lev(n) &
      flowPolicy(name(T1), m)‘self ≤ flowPolicy(name(T1), m)‘hp &
       flowPolicy(name(T1), m)‘self ≤ flowPolicy(name(T1), m)‘res &
        indis(Lobs, ρ, V, flowPolicyFld, lev)(s, s′ ) &
         up?(mcallS(V, me)(n, T1, g1, m, g2)(s)) &
          up?(mcallS(V, me)(n, T1, g1, m, g2)(s′ ))
     ⊃
     flowPolicy(name(T1), m)‘self ≤ Lobs ∨
      bottom?(g1(s)) ∨
       semNil?(down(g1(s))) ∨
        bottom?(g2(s)) ∨
         bottom?(me(loctype(valL(down(g1(s)))))(m)
                   (s‘1,
                    argStore(loctype(valL(down(g1(s)))), m, s‘1, down(g1(s)),
                             down(g2(s)))))
          ∨
          bottom?(g1(s′ )) ∨
           semNil?(down(g1(s′ ))) ∨
            bottom?(g2(s′ )) ∨
             bottom?(me(loctype(valL(down(g1(s′ )))))(m)
                       (s′ ‘1,
                        argStore(loctype(valL(down(g1(s′ )))), m, s′ ‘1, down(g1(s′ )),
                                 down(g2(s′ )))))
              ∨
              (∃ τ:
```

```
                    (ρ ⊆ τ) &
                     indis(Lobs, τ, V, flowPolicyFld, lev)
                           (updateVar(V)
                                      (n,
                                       (down(me(loctype(valL(down(g1(s)))))(m)
                                                (s‘1,
                                                 argStore
                                                 (loctype(valL(down(g1(s)))),
                                                  m,
                                                  s‘1,
                                                  down(g1(s)),
                                                  down(g2(s)))))‘1,
                                        s‘2),
                                      down(me(loctype(valL(down(g1(s)))))(m)
                                               (s‘1,
                                                argStore(loctype(valL(down(g1(s)))),
                                                         m,
                                                         s‘1,
                                                         down(g1(s)),
                                                         down(g2(s)))))‘2),
                            updateVar(V)
                                      (n,
                                       (down(me(loctype(valL(down(g1(s′ )))))(m)
                                                (s′ ‘1,
                                                 argStore
                                                 (loctype(valL(down(g1(s′ )))),
                                                  m,
                                                  s′ ‘1,
                                                  down(g1(s′ )),
                                                  down(g2(s′ )))))‘1,
                                        s′ ‘2),
                                      down(me(loctype(valL(down(g1(s′ )))))(m)
                                               (s′ ‘1,
                                                argStore(loctype(valL(down(g1(s′ )))),
                                                         m,
                                                         s′ ‘1,
                                                         down(g1(s′ )),
                                                         down
                                                         (g2
                                                          (s′ )))))‘2)))

  nonint_mcallS: LEMMA
    ∀ (Lobs: Level, V: Vxt, lev: Levels(V), ((n: (V‘dom) | n ≠ Self)),
       T1: (classT?), g1: SemExpr(V, T1),
       m: {m | definedMeth(name(T1))(m) & resType(name(T1))(m) ≤ V‘map(n)},
       g2: SemExpr(V, parType(name(T1))(m)),
       ((me: MethEnv | nonintMenv(me) & writeConfMenv(me)))):
      flowPolicy(name(T1), m)‘res ≤ lev(n) &
       flowPolicy(name(T1), m)‘self ≤ lev(n) &
        flowPolicy(name(T1), m)‘self ≤ flowPolicy(name(T1), m)‘hp &
         flowPolicy(name(T1), m)‘self ≤ flowPolicy(name(T1), m)‘res &
          readConf(flowPolicy(name(T1), m)‘self, V, T1, flowPolicyFld)(lev)(g1) &
           readConf(flowPolicy(name(T1), m)‘par, V, parType(name(T1))(m),
```

```
                    flowPolicyFld)
                   (lev)(g2)
    ⇒
    nonint(Lobs, V, lev)(mcallS(V, me)(n, T1, g1, m, g2))

nonint_ifelseS: LEMMA
  ∀ L, V, (g1, g2: SemCommand(V)), (lev: Levels(V)), (g: SemExpr(V, boolT)):
    nonint(L, V, lev)(g1) &
     nonint(L, V, lev)(g2) &
      (∃ L1, L2:
         L1 ≤ L2 &
          readConf(L1, V, boolT, flowPolicyFld)(lev)(g) &
           writeConf(L2, L2, V, flowPolicyFld, lev)(g1) &
            writeConf(L2, L2, V, flowPolicyFld, lev)(g2))
     ⇒ nonint(L, V, lev)(ifelseS(V)(g, g1, g2))

nonint_com: LEMMA
  ∀ L,
    V, S, (lev: Levels(V)), ((me: MethEnv | nonintMenv(me) & writeConfMenv(me))):
    up?(comSafe(V, lev, S)) ⇒
     nonint(L, V, lev)(comSem(V, me)(S))

nonintMenv_admiss: LEMMA
  ∀ (app: AscendingMenvs):
    (∀ j: nonintMenv(app(j))) ⇒ nonintMenv(lub(app))

nonint_com_to_meth: LEMMA
  ∀ L, C, (m: DeclaredMeth(C)), (g: SemCommand(bodyVxtFor(C, m))):
    nonint(L, bodyVxtFor(C, m), mbodyLevels(C, m))(g) ⇒
     nonint(L, C, m)(semComToMeth(C, m, g))

nonint_approx: LEMMA
  safe_CT ⇒ (∀ j, C: nonintMenv2(approxMethEnv(j), C))

nonint_approx2: LEMMA
  safe_CT ⇒ (∀ j: nonintMenv(approxMethEnv(j)))

nonint_CT: THEOREM safe_CT ⇒ nonintMenv(semCT)

END safeProps
```

```
% $Revision: 1.3 $ $Date: 2004/03/19 04:08:01 $

lift_props[T: TYPE]: THEORY
 BEGIN
  x: VAR T
  y: VAR lift[T]

  down_retracts_up: LEMMA down(up(x)) = x

  down_inverts_up: LEMMA FORALL (y: (up?[T])):  y = up(x) <=> down(y) = x

  AUTO_REWRITE+ down_retracts_up, down_inverts_up
 END lift_props
```

int_props: THEORY
BEGIN

$m$, $n$: VAR nat

max_above: LEMMA $\max(m, n) \geq m$ & $\max(m, n) \geq n$

END int_props

```
minProps: THEORY
 BEGIN

  IMPORTING min_nat[nat]

  T: TYPE = nat

  S, R: VAR (nonempty?[T])

  a, b, x: VAR T

  min_satisfies: LEMMA S(min(S))

  min_least: LEMMA S(x) ⇒ min(S) ≤ x

  min_gt: LEMMA (∀ x: x ≤ a ⇒ ¬ S(x)) ⇒ a < min(S)

  min_compareA: LEMMA
    ¬ S(0) & (∀ x: x > 0 & S(x) ⇒ R(x − 1)) ⇒
     (∀ x: S(x) ⇒ min(R) ≤ x − 1)

  min_compareC: LEMMA
    (∀ x: x ≤ min(R) ⇒ ¬ S(x)) ⇒ min(R) < min(S)

  min_compareB: LEMMA
    (S(x) ⇒ min(R) ≤ x − 1) =
     (x ≤ min(R) ⇒ ¬ S(x))

  min_compareD: LEMMA
    (∀ x: S(x) ⇒ min(R) ≤ x − 1) ⇒
     (∀ x: x ≤ min(R) ⇒ ¬ S(x))

  min_compare: LEMMA
    ¬ S(0) & (∀ x: x > 0 & S(x) ⇒ R(x − 1)) ⇒
     min(R) < min(S)

 END minProps
```

```
lattice: THEORY
 BEGIN

  Level: TYPE+

  ≤: (partial_order?[Level])

  botL, topL: Level

  glb, lub: [Level, Level → Level]

  L, L1, L2, L3, L4: VAR Level

  bot: AXIOM botL ≤ L

  top: AXIOM L ≤ topL

  glb_lower: AXIOM glb(L1, L2) ≤ L1 & glb(L1, L2) ≤ L2

  glb_greatest: AXIOM L ≤ L1 & L ≤ L2 ⇒ L ≤ glb(L1, L2)

  lub_upper: AXIOM L1 ≤ lub(L1, L2) & L2 ≤ lub(L1, L2)

  lub_least: AXIOM L1 ≤ L & L2 ≤ L ⇒ lub(L1, L2) ≤ L

  above_lub: LEMMA lub(L1, L2) ≤ L ⇒ L1 ≤ L & L2 ≤ L

  s_lub(LL1, LL2: lift[Level]): lift[Level] =
      IF bottom?(LL1) ∨ bottom?(LL2)
        THEN bottom
      ELSE up(lub(down(LL1), down(LL2)))
      ENDIF

  s_glb(LL1, LL2: lift[Level]): lift[Level] =
      IF bottom?(LL1) ∨ bottom?(LL2)
        THEN bottom
      ELSE up(glb(down(LL1), down(LL2)))
      ENDIF

  leq_lev_trans: LEMMA transitive?[Level](≤)

  not_above_glb: LEMMA
    ¬ glb(L1, L2) ≤ L ⇒ ¬ L1 ≤ L ∧ ¬ L2 ≤ L

  glb_lower_1: LEMMA glb(glb(L1, L2), glb(L3, L4)) ≤ L1

  glb_lower_2: LEMMA glb(glb(L1, L2), glb(L3, L4)) ≤ L2

  glb_lower_3: LEMMA glb(glb(L1, L2), glb(L3, L4)) ≤ L3

  glb_lower_4: LEMMA glb(glb(L1, L2), glb(L3, L4)) ≤ L4

 END lattice
```

# 4 PVS Status

Status from running prover on complete import chain from the main theory, safeProps. PVS 3.1 under Linux, 1.2Ghz Intel CPU, .5Gb real memory.

```
 Proof summary for theory confinement
    invar_subcl_field..................proved - complete   [shostak](  0.39 s)
    invar_subcl_meth...................proved - complete   [shostak](  0.33 s)
    errS_read_conf_TCC1................proved - complete   [shostak](  0.18 s)
    errS_read_conf.....................proved - complete   [shostak](  0.51 s)
    boolS_read_conf_TCC1...............proved - complete   [shostak](  0.13 s)
    boolS_read_conf....................proved - complete   [shostak](  0.51 s)
    nullS_read_conf_TCC1...............proved - complete   [shostak](  0.19 s)
    nullS_read_conf....................proved - complete   [shostak](  0.51 s)
    vblS_read_conf_TCC1................proved - complete   [shostak](  0.30 s)
    vblS_read_conf.....................proved - complete   [shostak]( 14.04 s)
    eqtest_read_conf_lem_TCC1..........proved - complete   [shostak](  0.24 s)
    eqtest_read_conf_lem_TCC2..........proved - complete   [shostak](  0.23 s)
    eqtest_read_conf_lem...............proved - complete   [shostak](  1.33 s)
    eqtestS_read_conf_TCC1.............proved - complete   [shostak](  0.93 s)
    eqtestS_read_conf..................proved - complete   [shostak](  2.04 s)
    fieldAccessS_read_conf_TCC1........proved - complete   [shostak](  0.30 s)
    fieldAccessS_read_conf.............proved - complete   [shostak](  3.40 s)
    typetestS_read_conf_TCC1...........proved - complete   [shostak](  0.16 s)
    typetestS_read_conf................proved - complete   [shostak](  1.40 s)
    castS_read_conf_TCC1...............proved - complete   [shostak](  0.27 s)
    castS_read_conf....................proved - complete   [shostak](  2.77 s)
    updateVar_write_conf...............proved - complete   [shostak](  1.93 s)
    extend_heap_write_conf_TCC1........proved - complete   [shostak](  0.09 s)
    extend_heap_write_conf_TCC2........proved - complete   [shostak](  3.25 s)
    extend_heap_write_conf_TCC3........proved - complete   [shostak](  2.59 s)
    extend_heap_write_conf_TCC4........proved - complete   [shostak](  1.14 s)
    extend_heap_write_conf_TCC5........proved - complete   [shostak](  0.32 s)
    extend_heap_write_conf.............proved - complete   [shostak](  2.89 s)
    updateField_write_conf.............proved - complete   [shostak](  3.64 s)
    mcallS_heap_effect_write_conf_TCC1..proved - complete   [shostak](  1.62 s)
    mcallS_heap_effect_write_conf_TCC2..proved - complete   [shostak]( 19.75 s)
    mcallS_heap_effect_write_conf_TCC3..proved - complete   [shostak](  7.34 s)
    mcallS_heap_effect_write_conf_TCC4..proved - complete   [shostak](  4.11 s)
    mcallS_heap_effect_write_conf_TCC5..proved - complete   [shostak](  4.22 s)
    mcallS_heap_effect_write_conf......proved - complete   [shostak](111.82 s)
    abortCom_write_conf................proved - complete   [shostak](  0.58 s)
    seqS_write_conf....................proved - complete   [shostak](  2.98 s)
    ifelseS_write_conf.................proved - complete   [shostak](  1.70 s)
    assignS_write_conf_TCC1............proved - complete   [shostak](  0.84 s)
    assignS_write_conf.................proved - complete   [shostak](  1.02 s)
    newS_write_conf....................proved - complete   [shostak](  6.47 s)
    fieldUpdateS_write_conf_TCC1.......proved - complete   [shostak](  0.93 s)
    fieldUpdateS_write_conf............proved - complete   [shostak]( 30.37 s)
    mcallS_write_conf_TCC1.............proved - complete   [shostak](  0.24 s)
    mcallS_write_conf..................proved - complete   [shostak]( 72.94 s)
    skipS_write_conf...................proved - complete   [shostak](  0.72 s)
    writeConfMeth_mono_TCC1............proved - complete   [shostak](  0.45 s)
```

```
    writeConfMeth_mono_TCC2............proved - complete   [shostak](  0.54 s)
    writeConfMeth_mono_TCC3............proved - complete   [shostak](  0.20 s)
    writeConfMeth_mono.................proved - complete   [shostak](  0.91 s)
    write_conf_abortMeth...............proved - complete   [shostak](  0.74 s)
    safe_exp_read_conf.................proved - complete   [shostak](  5.07 s)
    safe_com_write_conf................proved - complete   [shostak]( 11.06 s)
    safe_mbody_write_conf_TCC1.........proved - complete   [shostak](  0.41 s)
    safe_mbody_write_conf..............proved - complete   [shostak](  1.90 s)
    write_conf_admiss..................proved - complete   [shostak](  1.17 s)
    write_conf_approx..................proved - complete   [shostak](  1.97 s)
    safe_menv_write_conf...............proved - complete   [shostak](  0.39 s)
    Theory totals: 58 formulas, 58 attempted, 58 succeeded (338.47 s)

 Proof summary for theory indistinguishable
    idRel_typBij.......................proved - complete   [shostak]( 0.14 s)
    extendBij_typ......................proved - complete   [shostak]( 0.52 s)
    extendBij_subset...................proved - complete   [shostak]( 0.36 s)
    left_dom_id........................proved - complete   [shostak]( 0.31 s)
    rt_dom_id..........................proved - complete   [shostak]( 0.32 s)
    dom_rng_typBij.....................proved - complete   [shostak]( 0.43 s)
    typ_bij_comp.......................proved - complete   [shostak]( 0.18 s)
    dom_bij_comp.......................proved - complete   [shostak]( 0.35 s)
    rng_bij_comp.......................proved - complete   [shostak]( 0.35 s)
    typBij_type........................proved - complete   [shostak]( 0.32 s)
    subset_pointwise...................proved - complete   [shostak]( 0.55 s)
    indis_default......................proved - complete   [shostak]( 0.36 s)
    indis_TCC1.........................proved - complete   [shostak]( 0.07 s)
    indis_TCC2.........................proved - complete   [shostak]( 0.12 s)
    indis_TCC3.........................proved - complete   [shostak]( 1.36 s)
    indis_TCC4.........................proved - complete   [shostak]( 0.30 s)
    indis_TCC5.........................proved - complete   [shostak]( 0.26 s)
    writeConfMeth_TCC1.................proved - complete   [shostak]( 0.12 s)
    indis_trans_store..................proved - complete   [shostak]( 1.66 s)
    indis_trans_heap...................proved - complete   [shostak](11.10 s)
    idRel_comp_left....................proved - complete   [shostak]( 0.38 s)
    idRel_comp_rt......................proved - complete   [shostak]( 0.40 s)
    idRel_comp_subset..................proved - complete   [shostak]( 0.41 s)
    idRel_comp_left_subset.............proved - complete   [shostak]( 0.42 s)
    idRel_comp_rt_subset...............proved - complete   [shostak]( 0.43 s)
    indis_sym_id_store.................proved - complete   [shostak]( 0.72 s)
    indis_sym_id_heap..................proved - complete   [shostak](12.45 s)
    indis_sym_id_state.................proved - complete   [shostak](14.92 s)
    indis_id_refl_store................proved - complete   [shostak]( 0.89 s)
    indis_id_refl_heap.................proved - complete   [shostak]( 1.52 s)
    indis_id_refl_state................proved - complete   [shostak]( 0.55 s)
    indis_trans_store_ifelse...........proved - complete   [shostak](15.35 s)
    indis_trans_heap_ifelse............proved - complete   [shostak]( 0.40 s)
    indis_mono_val_TCC1................proved - complete   [shostak]( 0.06 s)
    indis_mono_val_TCC2................proved - complete   [shostak]( 0.06 s)
    indis_mono_val.....................proved - complete   [shostak]( 0.44 s)
    indis_antimono_val_TCC1............proved - complete   [shostak]( 3.62 s)
    indis_antimono_val_TCC2............proved - complete   [shostak]( 3.75 s)
    indis_antimono_val.................proved - complete   [shostak]( 0.42 s)
```

```
   indis_antimono_store................proved - complete   [shostak]( 0.52 s)
   indis_antimono_heap.................proved - complete   [shostak]( 2.40 s)
   indis_antimono_state................proved - complete   [shostak]( 0.37 s)
   indis_mono_bij_store................proved - complete   [shostak]( 3.44 s)
   indis_mono_result_TCC1..............proved - complete   [shostak]( 2.52 s)
   indis_mono_result_TCC2..............proved - complete   [shostak]( 1.80 s)
   indis_mono_result_TCC3..............proved - complete   [shostak]( 2.50 s)
   indis_mono_result...................proved - complete   [shostak]( 0.48 s)
   indis_antimono_rel_store............proved - complete   [shostak]( 3.71 s)
   indis_res_yields_indis_call_TCC1....proved - complete   [shostak](16.25 s)
   indis_res_yields_indis_call_TCC2....proved - complete   [shostak]( 0.28 s)
   indis_res_yields_indis_call.........proved - complete   [shostak]( 0.37 s)
   indis_extend_heap_pres_store_TCC1....proved - complete   [shostak]( 9.63 s)
   indis_extend_heap_pres_store_TCC2....proved - complete   [shostak]( 7.90 s)
   indis_extend_heap_pres_store........proved - complete   [shostak]( 0.36 s)
   read_conf_mono......................proved - complete   [shostak]( 0.62 s)
   write_conf_antimono.................proved - complete   [shostak]( 0.86 s)
   read_conf_mono_type_TCC1............proved - complete   [shostak]( 6.39 s)
   read_conf_mono_type.................proved - complete   [shostak]( 1.81 s)
   Theory totals: 58 formulas, 58 attempted, 58 succeeded (138.58 s)

 Proof summary for theory semCT
   initMbodyStore_TCC1.................proved - complete   [shostak]( 0.10 s)
   initMbodyStore_TCC2.................proved - complete   [shostak]( 0.22 s)
   initMbodyStore_TCC3.................proved - complete   [shostak]( 3.32 s)
   initMbodyState_TCC1.................proved - complete   [shostak]( 0.07 s)
   extractResult_TCC1..................proved - complete   [shostak]( 0.28 s)
   extractResult_TCC2..................proved - complete   [shostak]( 0.13 s)
   extractResult_TCC3..................proved - complete   [shostak]( 1.28 s)
   semComToMeth_type...................proved - complete   [shostak]( 1.13 s)
   mbodySem_TCC1.......................proved - complete   [shostak]( 0.13 s)
   mbodySem_TCC2.......................proved - complete   [shostak]( 0.59 s)
   inherit_defined_TCC1................proved - complete   [shostak]( 0.14 s)
   inherit_defined_TCC2................proved - complete   [shostak]( 0.43 s)
   inherit_defined_TCC3................proved - complete   [shostak]( 0.14 s)
   inherit_defined.....................proved - complete   [shostak]( 2.93 s)
   approxMethEnv_TCC1..................proved - complete   [shostak]( 0.08 s)
   approxMethEnv_TCC2..................proved - complete   [shostak]( 0.18 s)
   approxMethEnv_TCC3..................proved - complete   [shostak]( 0.80 s)
   approxMethEnv_TCC4..................proved - complete   [shostak]( 0.24 s)
   approxMethEnv_TCC5..................proved - complete   [shostak]( 0.45 s)
   approxMethEnv_TCC6..................proved - complete   [shostak](31.02 s)
   approxMethEnv_TCC7..................proved - complete   [shostak]( 0.45 s)
   approxMethEnv_TCC8..................proved - complete   [shostak](20.31 s)
   semComToMeth_mono_TCC1..............proved - complete   [shostak]( 0.22 s)
   semComToMeth_mono_TCC2..............proved - complete   [shostak]( 2.54 s)
   semComToMeth_mono_TCC3..............proved - complete   [shostak]( 7.20 s)
   semComToMeth_mono_TCC4..............proved - complete   [shostak]( 0.53 s)
   semComToMeth_mono...................proved - complete   [shostak]( 3.86 s)
   mbodySem_mono.......................proved - complete   [shostak]( 8.42 s)
   ascending_approx_step...............proved - complete   [shostak]( 8.02 s)
   ascending_approx....................proved - complete   [shostak]( 0.06 s)
   Theory totals: 30 formulas, 30 attempted, 30 succeeded (95.27 s)
```

```
Proof summary for theory comSemanticsProps
   seqS_mono...........................proved - complete   [shostak]( 7.26 s)
   ifelseS_mono........................proved - complete   [shostak](12.39 s)
   mcall_mono_TCC1.....................proved - complete   [shostak]( 1.59 s)
   mcall_mono_TCC2.....................proved - complete   [shostak]( 7.19 s)
   mcall_mono..........................proved - complete   [shostak](45.26 s)
   comSem_mono.........................proved - complete   [shostak](10.63 s)
   Theory totals: 6 formulas, 6 attempted, 6 succeeded (84.32 s)

Proof summary for theory comSemantics
   updateVar_TCC1......................proved - complete   [shostak]( 0.26 s)
   updateField_TCC1....................proved - complete   [shostak]( 0.21 s)
   updateField_TCC2....................proved - complete   [shostak]( 0.19 s)
   argStore_TCC1.......................proved - complete   [shostak]( 0.27 s)
   argStore_TCC2.......................proved - complete   [shostak]( 0.26 s)
   argStore_TCC3.......................proved - complete   [shostak]( 0.33 s)
   argStore_state......................proved - complete   [shostak]( 0.38 s)
   updateVar_inc_heap..................proved - complete   [shostak]( 0.08 s)
   updateVar_same_heap.................proved - complete   [shostak]( 0.08 s)
   updateField_same_dom................proved - complete   [shostak]( 0.08 s)
   assignS_TCC1........................proved - complete   [shostak]( 0.28 s)
   newS_TCC1...........................proved - complete   [shostak]( 0.11 s)
   newS_TCC2...........................proved - complete   [shostak]( 0.28 s)
   newS_TCC3...........................proved - complete   [shostak]( 1.16 s)
   newS_TCC4...........................proved - complete   [shostak]( 0.45 s)
   newS_TCC5...........................proved - complete   [shostak]( 0.32 s)
   fieldUpdateS_TCC1...................proved - complete   [shostak]( 0.12 s)
   fieldUpdateS_TCC2...................proved - complete   [shostak]( 0.45 s)
   fieldUpdateS_TCC3...................proved - complete   [shostak]( 0.47 s)
   fieldUpdateS_TCC4...................proved - complete   [shostak]( 0.17 s)
   fieldUpdateS_TCC5...................proved - complete   [shostak]( 0.30 s)
   fieldUpdateS_TCC6...................proved - complete   [shostak]( 0.43 s)
   mcallS_TCC1.........................proved - complete   [shostak]( 0.11 s)
   mcallS_TCC2.........................proved - complete   [shostak]( 0.21 s)
   mcallS_TCC3.........................proved - complete   [shostak]( 1.46 s)
   mcallS_TCC4.........................proved - complete   [shostak]( 0.66 s)
   mcallS_TCC5.........................proved - complete   [shostak]( 0.36 s)
   mcallS_TCC6.........................proved - complete   [shostak]( 2.04 s)
   mcallS_TCC7.........................proved - complete   [shostak]( 3.00 s)
   mcallS_TCC8.........................proved - complete   [shostak]( 1.04 s)
   mcallS_TCC9.........................proved - complete   [shostak]( 8.39 s)
   mcallS_TCC10........................proved - complete   [shostak](11.83 s)
   seqS_TCC1...........................proved - complete   [shostak]( 0.27 s)
   ifelseS_TCC1........................proved - complete   [shostak]( 0.29 s)
   ifelseS_TCC2........................proved - complete   [shostak]( 0.33 s)
   abortCom_type.......................proved - complete   [shostak]( 0.13 s)
   assignS_type........................proved - complete   [shostak]( 0.48 s)
   newS_type...........................proved - complete   [shostak]( 0.20 s)
   fieldS_type.........................proved - complete   [shostak]( 0.73 s)
   mcallS_type.........................proved - complete   [shostak](13.40 s)
   seqS_type...........................proved - complete   [shostak]( 0.71 s)
   ifelseS_type........................proved - complete   [shostak]( 1.43 s)
```

```
    skipS_type..........................proved - complete   [shostak]( 0.16 s)
    mcallS_parType_type_TCC1............proved - complete   [shostak]( 0.19 s)
    mcallS_parType_type_TCC2............proved - complete   [shostak]( 0.14 s)
    mcallS_parType_type_TCC3............proved - complete   [shostak]( 0.38 s)
    mcallS_parType_type_TCC4............proved - complete   [shostak]( 0.51 s)
    mcallS_parType_type_TCC5............proved - complete   [shostak]( 0.16 s)
    mcallS_parType_type.................proved - complete   [shostak]( 0.56 s)
    comSem_TCC1.........................proved - complete   [shostak]( 0.32 s)
    comSem_TCC2.........................proved - complete   [shostak]( 0.74 s)
    comSem_TCC3.........................proved - complete   [shostak]( 1.63 s)
    comSem_TCC4.........................proved - complete   [shostak]( 0.30 s)
    comSem_TCC5.........................proved - complete   [shostak]( 0.23 s)
    comSem_TCC6.........................proved - complete   [shostak]( 0.25 s)
    comSem_TCC7.........................proved - complete   [shostak]( 0.30 s)
    comSem_TCC8.........................proved - complete   [shostak]( 0.29 s)
    comSem_TCC9.........................proved - complete   [shostak]( 1.20 s)
    comSem_TCC10........................proved - complete   [shostak]( 0.29 s)
    comSem_TCC11........................proved - complete   [shostak]( 0.29 s)
    comSem_TCC12........................proved - complete   [shostak]( 0.34 s)
    comSem_TCC13........................proved - complete   [shostak](82.60 s)
    comSem_TCC14........................proved - complete   [shostak]( 9.37 s)
    comSem_TCC15........................proved - complete   [shostak]( 0.16 s)
    comSem_TCC16........................proved - complete   [shostak]( 0.23 s)
    comSem_TCC17........................proved - complete   [shostak]( 0.17 s)
    comSem_TCC18........................proved - complete   [shostak]( 0.17 s)
    Theory totals: 67 formulas, 67 attempted, 67 succeeded (154.73 s)

 Proof summary for theory expSemantics
    vblS_TCC1...........................proved - complete   [shostak]( 0.08 s)
    boolS_TCC1..........................proved - complete   [shostak]( 0.05 s)
    nullS_TCC1..........................proved - complete   [shostak]( 0.09 s)
    eqtestS_TCC1........................proved - complete   [shostak]( 0.06 s)
    eqtestS_TCC2........................proved - complete   [shostak]( 0.15 s)
    eqtestS_TCC3........................proved - complete   [shostak]( 0.10 s)
    fieldAccessS_TCC1...................proved - complete   [shostak]( 0.05 s)
    fieldAccessS_TCC2...................proved - complete   [shostak]( 0.24 s)
    fieldAccessS_TCC3...................proved - complete   [shostak]( 0.35 s)
    fieldAccessS_TCC4...................proved - complete   [shostak]( 0.62 s)
    fieldAccessS_TCC5...................proved - complete   [shostak]( 4.88 s)
    typetestS_TCC1......................proved - complete   [shostak]( 0.11 s)
    typetestS_TCC2......................proved - complete   [shostak]( 0.18 s)
    typetestS_TCC3......................proved - complete   [shostak]( 0.10 s)
    typetestS_TCC4......................proved - complete   [shostak]( 0.14 s)
    castS_TCC1..........................proved - complete   [shostak](10.59 s)
    errS_type_TCC1......................proved - complete   [shostak]( 0.18 s)
    errS_type...........................proved - complete   [shostak]( 0.08 s)
    vblS_type_TCC1......................proved - complete   [shostak]( 0.23 s)
    vblS_type...........................proved - complete   [shostak]( 0.17 s)
    boolS_type_TCC1.....................proved - complete   [shostak]( 0.19 s)
    boolS_type..........................proved - complete   [shostak]( 0.04 s)
    nullS_type_TCC1.....................proved - complete   [shostak]( 0.21 s)
    nullS_type..........................proved - complete   [shostak]( 0.14 s)
    castS_type_TCC1.....................proved - complete   [shostak]( 1.81 s)
```

```
   castS_type..........................proved - complete   [shostak]( 1.16 s)
   eqtestS_type_TCC1...................proved - complete   [shostak]( 0.21 s)
   eqtestS_type........................proved - complete   [shostak]( 0.08 s)
   fieldAccessS_type_TCC1..............proved - complete   [shostak]( 6.71 s)
   fieldAccessS_type...................proved - complete   [shostak]( 4.33 s)
   typetestS_type_TCC1.................proved - complete   [shostak]( 0.45 s)
   typetestS_type......................proved - complete   [shostak]( 0.14 s)
   expSem_TCC1.........................proved - complete   [shostak]( 0.17 s)
   expSem_TCC2.........................proved - complete   [shostak]( 0.12 s)
   expSem_TCC3.........................proved - complete   [shostak]( 0.45 s)
   expSem_TCC4.........................proved - complete   [shostak]( 0.44 s)
   expSem_TCC5.........................proved - complete   [shostak]( 0.11 s)
   expSem_TCC6.........................proved - complete   [shostak]( 0.19 s)
   expSem_TCC7.........................proved - complete   [shostak]( 0.17 s)
   expSem_TCC8.........................proved - complete   [shostak]( 0.20 s)
   expSem_TCC9.........................proved - complete   [shostak]( 0.88 s)
   expSem_TCC10........................proved - complete   [shostak]( 0.09 s)
   expSem_TCC11........................proved - complete   [shostak]( 0.17 s)
   expSem_TCC12........................proved - complete   [shostak]( 0.15 s)
   expSem_TCC13........................proved - complete   [shostak]( 2.80 s)
   expSem_TCC14........................proved - complete   [shostak]( 0.16 s)
   expSem_TCC15........................proved - complete   [shostak]( 0.18 s)
   expSem_TCC16........................proved - complete   [shostak]( 0.19 s)
   expSem_TCC17........................proved - complete   [shostak]( 1.04 s)
   expSem_TCC18........................proved - complete   [shostak]( 0.17 s)
   expSem_TCC19........................proved - complete   [shostak]( 0.18 s)
   expSem_TCC20........................proved - complete   [shostak]( 0.56 s)
   expSem_TCC21........................proved - complete   [shostak](25.02 s)
   Theory totals: 53 formulas, 53 attempted, 53 succeeded (67.36 s)

Proof summary for theory lift_props
   down_retracts_up....................proved - complete   [shostak](0.14 s)
   down_inverts_up.....................proved - complete   [shostak](0.33 s)
   Theory totals: 2 formulas, 2 attempted, 2 succeeded (0.47 s)

Proof summary for theory semanticDomains
   semBool.............................proved - complete   [shostak]( 0.06 s)
   semIt...............................proved - complete   [shostak]( 0.00 s)
   locType.............................proved - complete   [shostak]( 0.04 s)
   nilType.............................proved - complete   [shostak]( 0.04 s)
   loc_val_below.......................proved - complete   [shostak]( 0.11 s)
   loc_val_type........................proved - complete   [shostak]( 0.07 s)
   val_subsumption.....................proved - complete   [shostak]( 0.08 s)
   val_subsumptionT....................proved - complete   [shostak]( 0.19 s)
   closedStore_TCC1....................proved - complete   [shostak]( 0.11 s)
   heap_TCC1...........................proved - complete   [shostak]( 0.04 s)
   store_subsumpt_TCC1.................proved - complete   [shostak]( 0.23 s)
   store_subsumpt......................proved - complete   [shostak]( 0.45 s)
   state_subsumpt_TCC1.................proved - complete   [shostak]( 0.14 s)
   state_subsumpt......................proved - complete   [shostak]( 0.15 s)
   methPar_subsumpt_TCC1...............proved - complete   [shostak](19.92 s)
   methPar_subsumpt....................proved - complete   [shostak]( 0.18 s)
   val_Vxt_subsumpt_TCC1...............proved - complete   [shostak]( 0.10 s)
```

```
   val_Vxt_subsumpt....................proved - complete   [shostak]( 0.10 s)
   extend_heap_state...................proved - complete   [shostak](20.23 s)
   contValOfType_TCC1..................proved - complete   [shostak]( 0.18 s)
   contVal_subsumptionT................proved - complete   [shostak]( 0.12 s)
   bottom_closed.......................proved - complete   [shostak]( 0.05 s)
   nil_closed..........................proved - complete   [shostak]( 0.06 s)
   bool_closed.........................proved - complete   [shostak]( 0.05 s)
   store_contained.....................proved - complete   [shostak]( 0.09 s)
   sem_expr_subsumpt_TCC1..............proved - complete   [shostak]( 0.12 s)
   sem_expr_subsumpt...................proved - complete   [shostak]( 0.25 s)
   methResult_subcl_TCC1...............proved - complete   [shostak]( 0.04 s)
   methResult_subcl....................proved - complete   [shostak]( 0.12 s)
   semMeth_subsumpt_TCC1...............proved - complete   [shostak]( 2.31 s)
   semMeth_subsumpt_TCC2...............proved - complete   [shostak](17.36 s)
   semMeth_subsumpt....................proved - complete   [shostak](16.19 s)
   abortMeth_TCC1......................proved - complete   [shostak]( 0.11 s)
   default_TCC1........................proved - complete   [shostak]( 0.03 s)
   default_TCC2........................proved - complete   [shostak]( 0.03 s)
   default_TCC3........................proved - complete   [shostak]( 0.03 s)
   initObState_TCC1....................proved - complete   [shostak]( 0.04 s)
   initObState_TCC2....................proved - complete   [shostak]( 0.04 s)
   default_closed......................proved - complete   [shostak]( 0.17 s)
   init_closed.........................proved - complete   [shostak]( 0.20 s)
   val_subsumpt_x_TCC1.................proved - complete   [shostak]( 0.07 s)
   val_subsumpt_x......................proved - complete   [shostak]( 0.10 s)
   field_type_simple_TCC1..............proved - complete   [shostak]( 0.09 s)
   field_type_simple...................proved - complete   [shostak]( 0.16 s)
   field_type_V........................proved - complete   [shostak]( 0.11 s)
   loc_val_in_heap_TCC1................proved - complete   [shostak]( 0.20 s)
   loc_val_in_heap.....................proved - complete   [shostak]( 0.21 s)
   field_val_type_A_TCC1...............proved - complete   [shostak]( 0.13 s)
   field_val_type_A....................proved - complete   [shostak]( 0.33 s)
   field_val_type_TCC1.................proved - complete   [shostak]( 0.37 s)
   field_val_type_TCC2.................proved - complete   [shostak]( 0.15 s)
   field_val_type_TCC3.................proved - complete   [shostak]( 0.14 s)
   field_val_type......................proved - complete   [shostak]( 0.52 s)
   closedStore_inc_heap................proved - complete   [shostak]( 7.63 s)
   semCommand_result_subset............proved - complete   [shostak]( 0.85 s)
   semMeth_result_subset...............proved - complete   [shostak](11.40 s)
   Theory totals: 56 formulas, 56 attempted, 56 succeeded (102.29 s)

Proof summary for theory value
   loc_inj.............................proved - complete   [shostak](0.43 s)
   semBool_inj.........................proved - complete   [shostak](0.36 s)
   valL_inj............................proved - complete   [shostak](0.87 s)
   Theory totals: 3 formulas, 3 attempted, 3 succeeded (1.66 s)

Proof summary for theory orderDomains
   lub_TCC1............................proved - complete   [shostak]( 0.51 s)
   lub_TCC2............................proved - complete   [shostak](30.48 s)
   bot_leq_state.......................proved - complete   [shostak]( 0.06 s)
   abortMeth_bot.......................proved - complete   [shostak]( 0.50 s)
   botMethEnv_bot......................proved - complete   [shostak]( 0.50 s)
```

```
   asc_menvs_step......................proved - complete   [shostak]( 1.64 s)
   leq_com_reflexive...................proved - complete   [shostak]( 0.07 s)
   leq_meth_reflexive..................proved - complete   [shostak]( 0.05 s)
   characterize_menv...................proved - complete   [shostak]( 0.67 s)
   characterize_asc_menvs..............proved - complete   [shostak]( 0.21 s)
   characterize_lub_bot................proved - complete   [shostak]( 0.28 s)
   characterize_menv_lub...............proved - complete   [shostak]( 2.20 s)
   Theory totals: 12 formulas, 12 attempted, 12 succeeded (37.17 s)

Proof summary for theory wellformedCT
   varNotSRpar_TCC1.....................proved - complete   [shostak](0.07 s)
   bodyVxtFor_TCC1......................proved - complete   [shostak](0.06 s)
   bodyVxtFor_TCC2......................proved - complete   [shostak](0.11 s)
   bodyVxtFor_TCC3......................proved - complete   [shostak](0.14 s)
   bodyTypable_TCC1.....................proved - complete   [shostak](0.06 s)
   declared_is_defined..................proved - complete   [shostak](1.05 s)
   Theory totals: 6 formulas, 6 attempted, 6 succeeded (1.49 s)

Proof summary for theory typing
   expOK_TCC1...........................proved - complete   [shostak](0.11 s)
   expOK_TCC2...........................proved - complete   [shostak](0.17 s)
   expOK_TCC3...........................proved - complete   [shostak](0.10 s)
   expOK_TCC4...........................proved - complete   [shostak](0.14 s)
   expOK_TCC5...........................proved - complete   [shostak](0.14 s)
   comOK_TCC1...........................proved - complete   [shostak](0.12 s)
   comOK_TCC2...........................proved - complete   [shostak](0.18 s)
   comOK_TCC3...........................proved - complete   [shostak](0.09 s)
   comOK_TCC4...........................proved - complete   [shostak](0.15 s)
   var_in_V.............................proved - complete   [shostak](0.09 s)
   invert_vbl...........................proved - complete   [shostak](0.91 s)
   invert_bool..........................proved - complete   [shostak](0.38 s)
   invert_null..........................proved - complete   [shostak](0.44 s)
   invert_eqtest........................proved - complete   [shostak](0.35 s)
   invert_fieldAccess...................proved - complete   [shostak](0.67 s)
   invert_typetest......................proved - complete   [shostak](0.36 s)
   invert_cast..........................proved - complete   [shostak](0.20 s)
   invert_assign........................proved - complete   [shostak](0.36 s)
   invert_new...........................proved - complete   [shostak](0.40 s)
   invert_fieldUpdate...................proved - complete   [shostak](0.40 s)
   invert_mcall.........................proved - complete   [shostak](0.57 s)
   invert_ifelse........................proved - complete   [shostak](0.32 s)
   invert_seq...........................proved - complete   [shostak](0.29 s)
   Theory totals: 23 formulas, 23 attempted, 23 succeeded (6.94 s)

Proof summary for theory classtableSig
   stepsToObj_nonempty..................proved - complete   [shostak](0.08 s)
   step_to_obj..........................proved - complete   [shostak](0.38 s)
   steps_to_obj_B_TCC1..................proved - complete   [shostak](0.30 s)
   steps_to_obj_B.......................proved - complete   [shostak](0.36 s)
   steps_to_obj_A.......................proved - complete   [shostak](0.03 s)
   cdepth_TCC1..........................proved - complete   [shostak](1.67 s)
   cdepth_super.........................proved - complete   [shostak](0.19 s)
   subcl_reflexive......................proved - complete   [shostak](0.11 s)
```

```
   subcl_transitive.....................proved - complete   [shostak](0.51 s)
   subcl_reflexiveT.....................proved - complete   [shostak](0.08 s)
   subcl_transitiveT....................proved - complete   [shostak](0.13 s)
   object_top...........................proved - complete   [shostak](8.10 s)
   super_above..........................proved - complete   [shostak](0.07 s)
   class_below_class....................proved - complete   [shostak](0.06 s)
   class_below_class_lem................proved - complete   [shostak](0.11 s)
   def_subcl............................proved - complete   [shostak](0.05 s)
   def_super............................proved - complete   [shostak](0.06 s)
   defined_mtype_par_not_SR.............proved - complete   [shostak](0.17 s)
   resType_TCC1.........................proved - complete   [shostak](0.06 s)
   subclVxt_TCC1........................proved - complete   [shostak](0.06 s)
   subcl_methParVxt_TCC1................proved - complete   [shostak](0.11 s)
   subcl_methParVxt.....................proved - complete   [shostak](0.50 s)
   meth_subcl...........................proved - complete   [shostak](0.05 s)
   super_methParVxt_dom_TCC1............proved - complete   [shostak](4.11 s)
   super_methParVxt_dom.................proved - complete   [shostak](0.09 s)
   field_subcl..........................proved - complete   [shostak](0.12 s)
   field_subcl_V........................proved - complete   [shostak](0.11 s)
   field_subcl_Vf_TCC1..................proved - complete   [shostak](2.68 s)
   field_subcl_Vf.......................proved - complete   [shostak](0.08 s)
   resType_inh_TCC1.....................proved - complete   [shostak](5.90 s)
   resType_inh..........................proved - complete   [shostak](0.11 s)
   parType_inh..........................proved - complete   [shostak](0.11 s)
   Theory totals: 32 formulas, 32 attempted, 32 succeeded (26.55 s)

Proof summary for theory minProps
   min_satisfies........................proved - complete   [shostak](0.76 s)
   min_least............................proved - complete   [shostak](0.16 s)
   min_gt...............................proved - complete   [shostak](1.13 s)
   min_compareA_TCC1....................proved - complete   [shostak](0.13 s)
   min_compareA.........................proved - complete   [shostak](0.18 s)
   min_compareC.........................proved - complete   [shostak](0.05 s)
   min_compareB.........................proved - complete   [shostak](0.24 s)
   min_compareD.........................proved - complete   [shostak](0.18 s)
   min_compare..........................proved - complete   [shostak](0.18 s)
   Theory totals: 9 formulas, 9 attempted, 9 succeeded (3.01 s)

Proof summary for theory dty
   inj_classT...........................proved - complete   [shostak](0.38 s)
   Theory totals: 1 formulas, 1 attempted, 1 succeeded (0.38 s)

Proof summary for theory lang
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory lattice
   lesseqp_TCC1........................proved - complete   [shostak]( 0.16 s)
   above_lub...........................proved - complete   [shostak](10.01 s)
   s_lub_TCC1..........................proved - complete   [shostak]( 0.05 s)
   s_lub_TCC2..........................proved - complete   [shostak]( 0.04 s)
   leq_lev_trans.......................proved - complete   [shostak]( 1.04 s)
   not_above_glb.......................proved - complete   [shostak](14.18 s)
   glb_lower_1.........................proved - complete   [shostak]( 3.19 s)
```

```
   glb_lower_2.........................proved - complete   [shostak]( 2.61 s)
   glb_lower_3.........................proved - complete   [shostak]( 3.77 s)
   glb_lower_4.........................proved - complete   [shostak]( 2.56 s)
   Theory totals: 10 formulas, 10 attempted, 10 succeeded (37.61 s)

Proof summary for theory safe
   expSafe_TCC1.........................proved - complete   [shostak](0.19 s)
   expSafe_TCC2.........................proved - complete   [shostak](0.14 s)
   expSafe_TCC3.........................proved - complete   [shostak](0.10 s)
   expSafe_TCC4.........................proved - complete   [shostak](0.10 s)
   expSafe_TCC5.........................proved - complete   [shostak](0.28 s)
   expSafe_TCC6.........................proved - complete   [shostak](0.11 s)
   expSafe_TCC7.........................proved - complete   [shostak](0.10 s)
   comSafe_TCC1.........................proved - complete   [shostak](0.66 s)
   comSafe_TCC2.........................proved - complete   [shostak](0.31 s)
   comSafe_TCC3.........................proved - complete   [shostak](0.51 s)
   comSafe_TCC4.........................proved - complete   [shostak](0.09 s)
   comSafe_TCC5.........................proved - complete   [shostak](0.13 s)
   comSafe_TCC6.........................proved - complete   [shostak](0.62 s)
   comSafe_TCC7.........................proved - complete   [shostak](0.66 s)
   comSafe_TCC8.........................proved - complete   [shostak](0.11 s)
   comSafe_TCC9.........................proved - complete   [shostak](0.08 s)
   comSafe_TCC10........................proved - complete   [shostak](0.13 s)
   comSafe_TCC11........................proved - complete   [shostak](0.59 s)
   comSafe_TCC12........................proved - complete   [shostak](0.61 s)
   comSafe_TCC13........................proved - complete   [shostak](0.28 s)
   comSafe_TCC14........................proved - complete   [shostak](0.17 s)
   invert_safe_fieldAccess..............proved - complete   [shostak](0.09 s)
   invert_safe_eqtest...................proved - complete   [shostak](0.11 s)
   invert_safe_typetest.................proved - complete   [shostak](0.09 s)
   invert_safe_cast.....................proved - complete   [shostak](0.09 s)
   invert_safe_assign_TCC1..............proved - complete   [shostak](3.25 s)
   invert_safe_assign...................proved - complete   [shostak](0.10 s)
   invert_safe_fieldUpdate_TCC1.........proved - complete   [shostak](2.60 s)
   invert_safe_fieldUpdate..............proved - complete   [shostak](0.11 s)
   invert_safe_mcall_TCC1...............proved - complete   [shostak](1.06 s)
   invert_safe_mcall_TCC2...............proved - complete   [shostak](1.07 s)
   invert_safe_mcall....................proved - complete   [shostak](0.17 s)
   invert_safe_ifelse...................proved - complete   [shostak](0.12 s)
   invert_safe_seq......................proved - complete   [shostak](0.10 s)
   mbodyLevels_TCC1.....................proved - complete   [shostak](0.20 s)
   mSafe_TCC1...........................proved - complete   [shostak](0.13 s)
   mSafe_TCC2...........................proved - complete   [shostak](0.27 s)
   mSafe_TCC3...........................proved - complete   [shostak](0.26 s)
   safe_OK_exp..........................proved - complete   [shostak](0.17 s)
   safe_OK_com..........................proved - complete   [shostak](0.18 s)
   Theory totals: 40 formulas, 40 attempted, 40 succeeded (16.14 s)

Proof summary for theory int_props
   max_above............................proved - complete   [shostak](0.71 s)
   Theory totals: 1 formulas, 1 attempted, 1 succeeded (0.71 s)

Proof summary for theory safeProps
```

```
nonint_TCC1........................proved - complete   [shostak](  0.49 s)
nonint_TCC2........................proved - complete   [shostak](  0.47 s)
indis_mono_methParVxt_A_TCC1.......proved - complete   [shostak](  0.71 s)
indis_mono_methParVxt_A_TCC2.......proved - complete   [shostak](  0.66 s)
indis_mono_methParVxt_A_TCC3.......proved - complete   [shostak](  0.47 s)
indis_mono_methParVxt_A............proved - complete   [shostak](  7.09 s)
indis_antimono_methParVxt_A........proved - complete   [shostak](  6.72 s)
indis_mono_methParVxt_B_TCC1.......proved - complete   [shostak](  0.37 s)
indis_mono_methParVxt_B_TCC2.......proved - complete   [shostak](  0.34 s)
indis_mono_methParVxt_B............proved - complete   [shostak](  0.77 s)
indis_mono_methParVxt_TCC1.........proved - complete   [shostak](  0.39 s)
indis_mono_methParVxt_TCC2.........proved - complete   [shostak](  0.44 s)
indis_mono_methParVxt..............proved - complete   [shostak](  0.81 s)
indis_antimono_methParVxt..........proved - complete   [shostak](  0.90 s)
nonint_super_TCC1..................proved - complete   [shostak](  0.39 s)
nonint_super_TCC2..................proved - complete   [shostak](  0.92 s)
nonint_super.......................proved - complete   [shostak](  1.94 s)
nonint_updateVar_TCC1..............proved - complete   [shostak](  0.53 s)
nonint_updateVar_TCC2..............proved - complete   [shostak](  0.49 s)
nonint_updateVar...................proved - complete   [shostak](  0.54 s)
nonint_newS_ext_typ................proved - complete   [shostak](  1.58 s)
extend_heap_nonint_TCC1............proved - complete   [shostak](170.35 s)
extend_heap_nonint_TCC2............proved - complete   [shostak](  0.60 s)
extend_heap_nonint_TCC3............proved - complete   [shostak]( 20.99 s)
extend_heap_nonint_TCC4............proved - complete   [shostak](  1.33 s)
extend_heap_nonint_TCC5............proved - complete   [shostak](  1.22 s)
extend_heap_nonint_TCC6............proved - complete   [shostak]( 14.09 s)
extend_heap_nonint.................proved - complete   [shostak]( 14.17 s)
nonint_updateField_not_visible_TCC1.proved - complete   [shostak](100.65 s)
nonint_updateField_not_visible_TCC2.proved - complete   [shostak]( 45.42 s)
nonint_updateField_not_visible_TCC3.proved - complete   [shostak](  0.48 s)
nonint_updateField_not_visible_TCC4.proved - complete   [shostak](  0.49 s)
nonint_updateField_not_visible.....proved - complete   [shostak]( 20.24 s)
initMbodyState_indis_TCC1..........proved - complete   [shostak](  0.52 s)
initMbodyState_indis...............proved - complete   [shostak](  8.40 s)
extractResult_indis_TCC1...........proved - complete   [shostak](  1.77 s)
extractResult_indis_TCC2...........proved - complete   [shostak](  1.77 s)
extractResult_indis_TCC3...........proved - complete   [shostak](  1.63 s)
extractResult_indis................proved - complete   [shostak](  1.66 s)
nonint_abort.......................proved - complete   [shostak](  0.65 s)
nonint_skip........................proved - complete   [shostak](  0.55 s)
nonint_assignS_TCC1................proved - complete   [shostak](  1.69 s)
nonint_assignS.....................proved - complete   [shostak]( 11.77 s)
nonint_newS........................proved - complete   [shostak]( 23.70 s)
nonint_seqS........................proved - complete   [shostak](  4.04 s)
nonint_fieldUpdateS_TCC1...........proved - complete   [shostak](  2.60 s)
nonint_fieldUpdateS................proved - complete   [shostak]( 26.11 s)
nonint_mcallS_args_TCC1............proved - complete   [shostak](  1.41 s)
nonint_mcallS_args_TCC2............proved - complete   [shostak](  3.14 s)
nonint_mcallS_args_TCC3............proved - complete   [shostak](  1.28 s)
nonint_mcallS_args_TCC4............proved - complete   [shostak](  0.80 s)
nonint_mcallS_args_TCC5............proved - complete   [shostak](  0.87 s)
nonint_mcallS_args_TCC6............proved - complete   [shostak](  0.90 s)
```

```
nonint_mcallS_args_TCC7.............proved - complete   [shostak](  1.01 s)
nonint_mcallS_args_TCC8.............proved - complete   [shostak](  0.96 s)
nonint_mcallS_args_TCC9.............proved - complete   [shostak](  0.95 s)
nonint_mcallS_args_TCC10............proved - complete   [shostak](  0.82 s)
nonint_mcallS_args_TCC11............proved - complete   [shostak](  1.85 s)
nonint_mcallS_args_TCC12............proved - complete   [shostak](  1.42 s)
nonint_mcallS_args_TCC13............proved - complete   [shostak](  0.68 s)
nonint_mcallS_args..................proved - complete   [shostak]( 13.50 s)
nonint_mcallS_res_type_TCC1.........proved - complete   [shostak](  2.10 s)
nonint_mcallS_res_type_TCC2.........proved - complete   [shostak](  2.93 s)
nonint_mcallS_res_type_TCC3.........proved - complete   [shostak]( 20.86 s)
nonint_mcallS_res_type_TCC4.........proved - complete   [shostak](  2.97 s)
nonint_mcallS_res_type_TCC5.........proved - complete   [shostak](  3.10 s)
nonint_mcallS_res_type..............proved - complete   [shostak](  2.89 s)
nonint_mcallS_vis_TCC1..............proved - complete   [shostak](  0.56 s)
nonint_mcallS_vis_TCC2..............proved - complete   [shostak](  6.34 s)
nonint_mcallS_vis_TCC3..............proved - complete   [shostak](  1.99 s)
nonint_mcallS_vis_TCC4..............proved - complete   [shostak](  1.39 s)
nonint_mcallS_vis_TCC5..............proved - complete   [shostak](  0.57 s)
nonint_mcallS_vis_TCC6..............proved - complete   [shostak](  1.39 s)
nonint_mcallS_vis_TCC7..............proved - complete   [shostak](  0.96 s)
nonint_mcallS_vis_TCC8..............proved - complete   [shostak](  0.86 s)
nonint_mcallS_vis_TCC9..............proved - complete   [shostak](  2.75 s)
nonint_mcallS_vis_TCC10.............proved - complete   [shostak](  1.33 s)
nonint_mcallS_vis_TCC11.............proved - complete   [shostak](  1.55 s)
nonint_mcallS_vis_TCC12.............proved - complete   [shostak](  0.92 s)
nonint_mcallS_vis_TCC13.............proved - complete   [shostak](  1.49 s)
nonint_mcallS_vis_TCC14.............proved - complete   [shostak](  1.11 s)
nonint_mcallS_vis_TCC15.............proved - complete   [shostak](  0.74 s)
nonint_mcallS_vis_TCC16.............proved - complete   [shostak](  1.36 s)
nonint_mcallS_vis_TCC17.............proved - complete   [shostak](  1.56 s)
nonint_mcallS_vis_TCC18.............proved - complete   [shostak](  1.27 s)
nonint_mcallS_vis_TCC19.............proved - complete   [shostak](  1.35 s)
nonint_mcallS_vis_TCC20.............proved - complete   [shostak](  1.40 s)
nonint_mcallS_vis...................proved - complete   [shostak]( 31.63 s)
nonint_mcallS_invis_TCC1............proved - complete   [shostak](  0.91 s)
nonint_mcallS_invis_TCC2............proved - complete   [shostak](  4.64 s)
nonint_mcallS_invis_TCC3............proved - complete   [shostak](  1.95 s)
nonint_mcallS_invis_TCC4............proved - complete   [shostak](  1.35 s)
nonint_mcallS_invis_TCC5............proved - complete   [shostak](  0.57 s)
nonint_mcallS_invis_TCC6............proved - complete   [shostak](  1.36 s)
nonint_mcallS_invis_TCC7............proved - complete   [shostak](  0.93 s)
nonint_mcallS_invis_TCC8............proved - complete   [shostak](  0.84 s)
nonint_mcallS_invis_TCC9............proved - complete   [shostak](  2.11 s)
nonint_mcallS_invis_TCC10...........proved - complete   [shostak](  1.19 s)
nonint_mcallS_invis_TCC11...........proved - complete   [shostak](  1.33 s)
nonint_mcallS_invis_TCC12...........proved - complete   [shostak](  0.90 s)
nonint_mcallS_invis_TCC13...........proved - complete   [shostak](  2.23 s)
nonint_mcallS_invis_TCC14...........proved - complete   [shostak](  1.03 s)
nonint_mcallS_invis_TCC15...........proved - complete   [shostak](  1.22 s)
nonint_mcallS_invis_TCC16...........proved - complete   [shostak](  1.38 s)
nonint_mcallS_invis_TCC17...........proved - complete   [shostak](  1.55 s)
nonint_mcallS_invis_TCC18...........proved - complete   [shostak](  1.27 s)
```

```
    nonint_mcallS_invis_TCC19..........proved - complete   [shostak](  1.29 s)
    nonint_mcallS_invis_TCC20..........proved - complete   [shostak](  1.36 s)
    nonint_mcallS_invis................proved - complete   [shostak]( 38.34 s)
    nonint_mcallS_TCC1.................proved - complete   [shostak](  0.54 s)
    nonint_mcallS......................proved - complete   [shostak]( 60.26 s)
    nonint_ifelseS.....................proved - complete   [shostak](  4.54 s)
    nonint_com.........................proved - complete   [shostak](  8.75 s)
    nonintMenv_admiss..................proved - complete   [shostak](  0.94 s)
    nonint_com_to_meth_TCC1............proved - complete   [shostak](  0.48 s)
    nonint_com_to_meth_TCC2............proved - complete   [shostak](  0.36 s)
    nonint_com_to_meth.................proved - complete   [shostak](  1.78 s)
    nonint_approx......................proved - complete   [shostak](  4.62 s)
    nonint_approx2.....................proved - complete   [shostak](  0.33 s)
    nonint_CT..........................proved - complete   [shostak](  0.32 s)
    Theory totals: 120 formulas, 120 attempted, 120 succeeded (781.54 s)

Grand Totals: 587 proofs, 587 attempted, 587 succeeded (1894.69 s)
```

### Acknowledgements

Natarajan Shankar and Patrick Lincoln arranged a Visiting Fellowship at SRI International for the month of September, 2003, during which time the author learned PVS and more than half of this work was carried out. Their Formal Methods group was very hospitable. Harold Reuß's elegant domain theoretic work [] was a nice example for my initial study. Shankar, Sam Owre, and Bruno Dutetre were particularly generous with explanations and advice (and Bruno taught me that the way to smash is to `(smash)`). Walkthroughs by the whole group, with John Rushby guiding me in fluent emacs-keystroke, were very helpful and encouraging.

Cesar Munoz helped with strategies for TCCs.

## References

[AG98] Ken Arnold and James Gosling. *The Java Programming Language, second edition.* Addison-Wesley, 1998.

[BN02] Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in a Java-like language. In *15th IEEE Computer Security Foundations Workshop*, pages 253–270, 2002.

[BN03a] Anindya Banerjee and David A. Naumann. Stack-based access control for secure information flow. *Journal of Functional Programming*, 2003. accepted for special issue on Language Based Security.

[BN03b] Anindya Banerjee and David A. Naumann. Using access control for secure information flow in a Java-like language. In *CSFW*, pages 155–169. IEEE Computer Society Press, 2003.

[Gon99] Li Gong. *Inside Java 2 Platform Security.* Addison-Wesley, 1999.

[IPW01] Atsushi Igarashi, Benjamin Pierce, and Philip Wadler. Featherweight Java: A minimal core calculus for Java and GJ. *ACM Transactions on Programming Languages and Systems*, 23(3):396–459, May 2001.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[Str03] Martin Strecker. Formal analysis of an information flow type system for MicroJava (extended version). Technical report, Technische Universität München, July 2003.

[VSI96] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.