Java Type Safety

March 7, 2002

1 ASM State

The state consists of many components scattered throughout the book. The components of the state are all collected here for easy reference.

N	State component	Type	Comment
1	*thread*	*Thread*	the current (active) thread
2	*meth*	*Class/MSig*	the current executing method (in the current thread)
3	*restbody*	*Pos* → *Phrase*	body of current method
4	*pos*	*Pos*	index to current subexpression of interest within body
5	*locals*	*Loc* → *Val*	environment holding values for local variables
6	*globals*	*Class/Field* → *Val*	environment holding values for static fields
7	*frames*	*Frame**	Stack of frames *Frame* = (*Class/MSig*, *Phrase*, *Pos*, *Locals*)
8	*heap*	*Ref* → *Val*	Heap
9	*classState*	*Class* → *ClassState*	State of each loaded class: *Linked*, *InProgress*, *Initialized*, or *Unusable*
10	*cont*	*Thread* → (*Frame**, *Frame*)	continuation (stack) of each non-active thread
11	*sync*	*Thread* → *Ref**	for each thread, the stack of objects whose locks were grabbed by the thread
12	*locks*	*Ref* → *Nat*	for each object, the number of times the lock has been grabbed (by some thread)
13	*waitSet*	*Ref* → *Powerset*(*Thread*)	for each object, the set of threads waiting to access this object
14	*exec*	*Thread* → *ThreadState*	for each thread, its state: *NotStarted*, *Active*, *Synchronizing*, *Waiting*, *Notified*, or *Dead*
15	*syncObj*	*Thread* → *Ref*	for each (synchronizing) thread, the object whose lock the thread is waiting to acquire
16	*waitObj*	*Thread* → *Ref*	for each (waiting) thread, the object on which it is waiting
17	*interruptedFlag*	*Thread* → *Bool*	has this thread been interrupted?
18	*initWait*	*Class* → *Powerset*(*Thread*)	for each class, the set of threads waiting for the initialization of the class to complete
19	*initThread*	*Class* → *Thread*	for each class, the thread that is initializing it

2 Type Safety

First it is convenient to add the time as a subscript to the dynamic functions. So locals_n refers to the value of *locals* in the n^{th} state (after n steps). Second we may also index frames_n, meth_n, $\mathit{restbody}_n$, pos_n, locals_n with a thread q frames^q_n, meth^q_n, $\mathit{restbody}^q_n$, pos^q_n, locals^q_n. If the thread q is the current thread in state n, then the indexed components correspond to the state components. If the thread q is not the current thread in state n, then the indexed components correspond to the components pushed in the continuation $\mathit{cont}_n(q)$.

A state n consisting of the 19 components above is *state-safe* if:

- For each thread q, each frame of the thread is *frame-safe*.
- For each thread q, each chain of frames on the stack for the thread is *chain-safe*.
- The following hold:
  - The source program typechecks and satisfies the rules of definite assignment and all other restrictions required by the Java Language Specification.
  - The declared type of a static field agrees with its value in $\mathit{globals}_n$. The declared type of an instance field agrees with its value in the allocated object in heap_n.
  - If any component of the state mentions a reference r (for example, r is the value of a static field, or r is the value of a local variable in the state), then r is in the domain of heap_n (it is not a dangling reference).
  - If heap_n contains an instance of class A, then class A is not abstract, and the heap-allocated instance includes values for all the fields declared in the class and its superclasses.

A frame $(\mathit{meth}, \mathit{resbody}, \mathit{pos}, \mathit{locals})$ is *frame-safe* if:

- $\mathit{before}(\mathit{pos}) \subseteq \mathit{dom}(\mathit{locals})$: any variable that, according to the rules of definite assignment, is definitely assigned before the current position, must be in the domain of the environment.
- If $\mathit{restbody}/\alpha$ is a value of type B, then B is run-time compatible with the compile-time type of position α.
- If *pos* is in the scope of a local variable declaration of variable x of type A and $x \in \mathit{dom}(\mathit{locals})$ then $\mathit{locals}(x)$ is a value of type B that is run-time compatible with A.
- and many more ...

A chain of frames is *chain-safe* if:

- The return type of one frame is consistent with the type expected by the parent frame.
- and more ...

Outline of the Proof of Type Safety

Just like the proof of type safety for MinML, the proof is by induction of the number of steps in the run of the ASM. One needs to show that the initial state is state-safe (base case), and that a safe state is either a final state or it can be updated to another state which is also state-safe.

Here we outline just one case. Consider a state where the phrase at the current position is a use of a local variable. This is not a final state; the transition rules in Fig. 3.2 show that this state can be rewritten to a new state in which the use of the local variable at the current position is replaced by its value in the environment *locals*. To be able to make this transition we must know that the variable is in the domain of *locals* and to finish the proof in this case, we must show that the resulting state is still state-safe.

To argue that the variable must be in the domain of *locals* we proceed as follows. Given that the current state is state-safe we know that $\mathit{before}(\mathit{pos}) \subseteq \mathit{dom}(\mathit{locals})$. Because the program satisfies the rules of definite assignment, the variable used must have been definitely assigned, and hence must be in the set $\mathit{before}(\mathit{pos})$. This means that it is also in the domain of *locals*.

To show that the resulting is state-safe, it suffices to note that the use of the variable must occur in the scope of a declaration, and hence we are guaranteed that the value in *locals* is run-time compatible with the type of the declaration.