1 Module NEW-INSTANCE

1.1 Background

In this subsection we present the fragment of configuration used by runtime method invocation. The figure below contains the cells and their sorts.

$$\left\langle \begin{array}{c} \left\langle \left\langle \begin{array}{c} \langle K\rangle_{\text{k}} \\ \langle\langle Map\,[Id \mapsto Int]\rangle_{\text{env}}\;\; \langle ClassType\rangle_{\text{crntClass}}\;\; \langle Int\rangle_{\text{location}}\rangle_{\text{methodContext}} \end{array} \right\rangle_{\text{thread}*} \right\rangle_{\text{threads}} \\ \left\langle \left\langle \begin{array}{c} \langle ClassType\rangle_{\text{classType}}\;\; \langle ClassType\rangle_{\text{enclosingClass}} \\ \langle ClassType\rangle_{\text{extends}}\;\; \langle List[FieldDec]\rangle_{\text{instanceFields}} \end{array} \right\rangle_{\text{class}*} \right\rangle_{\text{classes}} \\ \langle Map\,[Int \mapsto TypedVal]\rangle_{\text{store}}\;\; \langle Int\rangle_{\text{nextLoc}} \\ \left\langle \left\langle \begin{array}{c} \langle Int\rangle_{\text{objectId}}\;\; \langle ClassType\rangle_{\text{objectType}} \\ \left\langle \begin{array}{c} \langle ClassType\rangle_{\text{layerClass}}\;\; \langle Map\,[Id \mapsto Int]\rangle_{\text{layerEnv}} \\ \langle K\rangle_{\text{layerEnclosingObject}} \end{array} \right\rangle_{\text{layer}*} \end{array} \right\rangle_{\text{object}*} \right\rangle_{\text{objectStore}} \end{array} \right\rangle_{\top}$$

The cell $\langle\rangle_{\text{k}}$ stores the current computation. Inside $\langle\rangle_{\text{env}}$ we store the local environment – a map from variable names to their locations in the store. The cell $\langle\rangle_{\text{methodContext}}$ store information about the current object – the one accessible through the keyword this. Both $\langle\rangle_{\text{env}}$ and $\langle\rangle_{\text{methodContext}}$ play a special role in object instantiation.

The cell $\langle\rangle_{\text{class}}$ contains various sub-cells holding the content of that class. The first cell in $\langle\rangle_{\text{classType}}$ of sort ClassType that holds the fully qualified class name. This cell is a unique identifier of a class, and is used as a key to access other cells inside a $\langle\rangle_{\text{class}}$. Next relevant cells inside $\langle\rangle_{\text{class}}$ are $\langle\rangle_{\text{enclosingClass}}$ - the directly enclosing class in case this class is an inner class. The vase class is stored inside $\langle\rangle_{\text{extends}}$ and the list of declarations of instance fields without identifiers is stored in $\langle\rangle_{\text{instanceFields}}$.

The next two cells are related to the store. The cell $\langle\rangle_{\text{store}}$ have a central role in the semantics – it is the map from object locations (values in the cell $\langle\rangle_{\text{env}}$) to their actual typed values. The cell $\langle\rangle_{\text{nextLoc}}$ is the counter of store locations.

The remaining big group of cells – $\langle\rangle_{\text{objectStore}}$ contains the inner structure of objects. The $\langle\rangle_{\text{objectId}}$ is an unique identifier of the object. Every reference to this object in the store is a reference to this id. Inside $\langle\rangle_{\text{objectType}}$ is the actual runtime type of the object. Next we have a list of $\langle\rangle_{\text{layer}}$ cells, each of them representing an inheritance layer of the object. Starting from class Object and ending with the actual object type. Inside each layer $\langle\rangle_{\text{layerClass}}$ stores its associated class, $\langle\rangle_{\text{layerEnv}}$ – the fields and $\langle\rangle_{\text{layerEnclosingObject}}$ – the enclosing object, in the case when $\langle\rangle_{\text{layerClass}}$ is a non-static inner class. The complex rules for Java inner classes allow each layer to have its distinctive enclosing object, and we have tests that specifically target this requirement.

1.2 New instance creation

When a new instance creation expression reaches the top of computation, first it is normalized to a standard form. If it is an unqualified expression, an empty qualifier is added. Second, if the class to be instantiated is a simple name, it have to be converted to a fully qualified class name. At this stage this could only happen for true inner classes, and the fully qualified name is computed by concatenating the type of the qualifier and the class simple name, by the rule below.

RULE QUALIFIED-NEW-INSTANCE-RESOLVE-CLASS

$$Qual\ .\ \texttt{new}\ \frac{Name}{\texttt{getClassType}\ (\ \texttt{toPackage}\ (\ \texttt{typeOf}\ (Qual)),\ Name)}(\_)$$

After the new instance expression have been normalized, the qualifier and the arguments are brought to the top of computation by the strictness rules and evaluated. Qualifier is evaluated first, and arguments are evaluated left-to-right according to JLS.

When all the subexpressions of new have been evaluated, the main rule for new could apply. This rule touches a large number of cells, that will be explained next. First the current value of the counter inside $\langle\rangle_{\mathsf{nextLoc}}$ is used as the location of the newly created object. The counter is incremented for the next use. Inside $\langle\rangle_{\mathsf{objectStore}}$ a new cell $\langle\rangle_{\mathsf{object}}$ is created for the new object. For now it have just two sub-cells specified – $\langle\rangle_{\mathsf{objectId}}$ and $\langle\rangle_{\mathsf{objectType}}$, and no layers. Curiously we don't have to specify neither $\langle\rangle_{\mathsf{object}}$ nor $\langle\rangle_{\mathsf{objectStore}}$ cells explicitly here, we have to specify just the cells inside them that are modified. The capability to ignore surrounding cells when they can be automatically inferred is called configuration abstraction, another K feature[?]. In the cell $\langle\rangle_{\mathsf{store}}$ a new entry is created with key being L and value - a reference to the newly created object in $\langle\rangle_{\mathsf{object}}$. The content of $\langle\rangle_{\mathsf{methodContext}}$ is reset to a default state. This default state is required by rules that are applied next.

Inside $\langle\rangle_{\mathsf{k}}$ the new instance expression is rewritten into a sequence of computations that will be executed by the following rules. The auxiliary function staticInit() triggers static initialization of the instantiated class, in case it was not triggered earlier. Next, the function create() populates the layers of the object inside $\langle\rangle_{\mathsf{object}}$ This also includes allocation of all instance fields, and their initialization to the default value. Field initializers are not executed yet. The function setEncloser() sets the enclosing object for the current class, if the current class is an inner class. If some of the base classes are also inner classes, the encloser for their respective $\langle\rangle_{\mathsf{layer}}$ will be set as part of constructor invocation.

The next term in the computation (the one starting with typedLookup(L)) might look a bit weird, but it is in fact the invocation of the constructor. This term represents a mix of Java syntax for method invocation and auxiliary functions defined inside K-Java. It illustrates, among others, the power of K parser. Now, after all memory allocation procedures have been completed, it is the right time for it to be invoked. Preprocessing semantics transforms all constructors into plain methods. The function typedLookup(L) is evaluated into the object stored at the location L, that will serve as a qualifier for constructor invocation. The function getConsName() converts the class name into the name of the constructor method. What remains is plain Java syntax for method invocation.

The last two terms bring computation to the state required to continue execution. Function restoreMetho-Context() restores $\langle\rangle_{\mathsf{methodContext}}$ to the the state before object creation. The last term is the result value of the object instantiation expression.

RULE QUALIFIED-NEW-INSTANCE

$$\left\langle \frac{Qual\ \texttt{.}\ \texttt{new}\ Class(Args)}{\begin{array}{c}\texttt{staticInit}\ (Class) \curvearrowright \texttt{create}\ (Class) \\ \curvearrowright \texttt{setEncloser}\ (\ \texttt{typedLookup}\ (L), Class, Qual) \\ \curvearrowright \texttt{typedLookup}\ (L)\ \texttt{.}\ \texttt{getConsName}\ (Class)(Args)\ \texttt{;} \\ \curvearrowright \texttt{restoreMethContext}\ (MethContext) \curvearrowright \texttt{typedLookup}\ (L)\end{array}} \cdots \right\rangle_{\mathsf{k}} \left\langle \cdots \frac{\cdot Map}{L \mapsto \texttt{objectRef}\ (L, Class)\ \texttt{::}\ Class} \cdots \right\rangle_{\mathsf{sto}}$$

$$\left\langle \frac{L}{L +_{Int} 1} \right\rangle_{\mathsf{nextLoc}} \frac{\cdot Bag}{\langle \langle L \rangle_{\mathsf{objectId}}\ \langle Class \rangle_{\mathsf{objectType}} \rangle_{\mathsf{object}}} \left\langle \frac{MethContext}{\langle \cdot Map \rangle_{\mathsf{env}}\ \langle \cdot K \rangle_{\mathsf{crntClass}}\ \langle L \rangle_{\mathsf{location}}} \right\rangle_{\mathsf{methodContext}}$$

SYNTAX $K ::= \texttt{create}\ (ClassType)$

RULE CREATE

$$\left\langle \frac{\texttt{create}\ (Class)}{\texttt{create}\ (BaseClass) \curvearrowright \texttt{setCrntClass}\ (Class) \curvearrowright FieldDecs \curvearrowright \texttt{addEnvLayer}} \cdots \right\rangle_{\mathsf{k}} \quad \langle Class \rangle_{\mathsf{classType}}$$

$$\langle BaseClass \rangle_{\mathsf{extends}}\ \langle FieldDecs \rangle_{\mathsf{instanceFields}}$$

[structural]

RULE CREATE-EMPTY-DISCARD

$$\frac{\texttt{create}\,(\cdot_{K})}{\cdot_{K}}$$

[structural]

SYNTAX $K ::= \texttt{setCrntClass}\,(\mathit{ClassType})$

RULE SETCRNTCLASS

$$\left\langle \frac{\texttt{setCrntClass}\,(\mathit{Class})}{\cdot_{K}} \cdots \right\rangle_{\mathsf{k}} \left\langle \frac{\_}{\mathit{Class}} \right\rangle_{\mathsf{crntClass}}$$

[structural]

SYNTAX $K ::= \texttt{addEnvLayer}$

RULE ADDENVLAYER

$$\left\langle \frac{\texttt{addEnvLayer}}{\cdot_{K}} \cdots \right\rangle_{\mathsf{k}} \quad \left\langle \frac{\mathit{Env}}{\cdot_{Map}} \right\rangle_{\mathsf{env}} \quad \langle \mathit{Class} \rangle_{\mathsf{crntClass}} \quad \langle \mathit{OId} \rangle_{\mathsf{location}}$$

$$\left\langle \langle \mathit{OId} \rangle_{\mathsf{objectId}} \ \frac{\cdot_{Bag}}{\langle \langle \mathit{Class} \rangle_{\mathsf{layerClass}} \ \langle \mathit{Env} \rangle_{\mathsf{layerEnv}} \cdots \rangle_{\mathsf{layer}}} \cdots \right\rangle_{\mathsf{object}}$$

[structural]

Sets the enclosing object for a given object.

SYNTAX $K ::= \texttt{setEncloser}\,(K, \mathit{ClassType}, K)$ [strict(1,3)]

RULE SETENCLOSER-VALUE

$$\left\langle \frac{\texttt{setEncloser}\,(\,\texttt{objectRef}\,(\mathit{OId}, \_)\ \texttt{::}\ \_, \mathit{Class}, \mathit{EncloserVal}\ \texttt{::}\ \_)}{\cdot_{K}} \cdots \right\rangle_{\mathsf{k}} \quad \langle \mathit{OId} \rangle_{\mathsf{objectId}} \quad \langle \mathit{Class} \rangle_{\mathsf{layerClass}}$$

$$\left\langle \frac{\_}{\mathit{EncloserVal}\ \texttt{::}\ \mathit{EncloserClass}} \right\rangle_{\mathsf{layerEnclosingObject}} \quad \langle \mathit{Class} \rangle_{\mathsf{classType}} \quad \langle \mathit{EncloserClass} \rangle_{\mathsf{enclosingClass}}$$

RULE SETENCLOSER-NOVALUE

$$\frac{\texttt{setEncloser}\,(\_, \_, \cdot_{K})}{\cdot_{K}}$$