This is a draft of a cheat paper for KeY. Take with care.

Java Card DL Syntax

Formulas

The table below relates KeY syntax with the standard textbook syntax. The greek letters φ, ϕ, ψ denote Java DL formulas.

KeY	Textbook	Remarks
true, false	tt, ff	truth values
`p`($\mathtt{t_1}, \ldots, \mathtt{t_n}$)	$p(t_1, \ldots, t_n)$	atomic formula
`!` ϕ	$\neg\phi$	negation
ϕ `&` ψ	$\phi \wedge \psi$	conjunction
$\phi \mid \psi$	$\phi \vee \psi$	disjunction
ϕ `->` ψ	$\phi \rightarrow \psi$	implication
ϕ `<->` ψ	$\phi \leftrightarrow \psi$	equivalence
\if (φ) \then (ϕ) \else (ψ)	—	conditional formula evaluates to ϕ if φ holds to ψ otherwise
\forall T x; ϕ	$\forall x{:}T.\,\phi$	universal quantification over elements of type T
\exists T x; ϕ	$\exists x{:}T.\,\phi$	existential quantification over elements of type T
$\{\mathcal{U}\}\phi$	$\{\mathcal{U}\}\phi$	update application (see Sect. *Updates*)
`\<{`p`}\>` ϕ	$\langle \mathtt{p} \rangle \phi$	diamond modality (total correctness) with statement list p and formula ϕ
`\[{`p`}\]` ϕ	$[p]\phi$	box modality (partial correctness) with statement list p and formula ϕ

Reserved predicate symbols

Some predefined predicates:

Symbol	Remarks
$\cdot \doteq \cdot$	equality
$\cdot < \cdot$, $\cdot <= \cdot$, $\cdot > \cdot$, $\cdot >= \cdot$	inequalities
`inReachableState`	true in states reachable by a Java program
`arrayStoreValid`(ar,el)	true if the element el can be stored in the array referenced by ar without causing an `ArrayStoreException`

Terms

Terms are sorted and recursively defined as usual.

KeY	Textbook	Remarks
`f`($\mathtt{t_1}, \ldots, \mathtt{t_n}$)	$f(t_1, \ldots, t_n)$	f function symbol, $t_1, \ldots, t_n$ terms of compatible sort
$\{\mathcal{U}\}t$	$\{\mathcal{U}\}t$	update application (see Sect. *Updates*)
\if (φ) \then (t_1) \else (t_2)	—	conditional term evaluates to t_1 if φ holds, to t_2 otherwise
`(`T`)`t	—	cast term t to type T

Reserved function symbols

Arithmetics

Prefix	In-/Postfix	Remark
add($\cdot$,$\cdot$)	$\cdot$ `+` $\cdot$	addition on $\mathbb{Z}$
sub($\cdot$,$\cdot$)	$\cdot$ `-` $\cdot$	substraction on $\mathbb{Z}$
mul($\cdot$,$\cdot$)	$\cdot$ `*` $\cdot$	multiplication on $\mathbb{Z}$
div($\cdot$,$\cdot$)	$\cdot$ `/` $\cdot$	division on $\mathbb{Z}$
mod($\cdot$,$\cdot$)	$\cdot$ `%` $\cdot$	modulo on $\mathbb{Z}$
jdiv($\cdot$,$\cdot$)		Java division (rounds towards 0), but on $\mathbb{Z}$
jmod($\cdot$,$\cdot$)		Java modulo, but on $\mathbb{Z}$
divJint($\cdot$,$\cdot$)		Java division resp. `int` bounds

Attributes and Arrays

Prefix	In-/Postfix	Remark
	$o.a$`@(T)`	attribute access term (access attribute a declared in T of object o); `@( · )` can be omitted if no hiding
	ar`[`idx`]`	array access term evaluating to the element stored at index idx in array ar

Other Interpreted Function Symbols

Prefix	Remark
`null`	Javas `null` constant (only element of type `Null`)
`TRUE`, `FALSE`	constants of type `boolean` with the obvious interpretation
T`::instance(`o`)`	boolean typed function evaluating to `TRUE` if o is an instance of type T

Updates

The general form of a single quantified update in KeY

$$\underbrace{\backslash\text{for } T\ x\texttt{;}_{opt}\ \backslash\text{if}(\varphi)_{opt} loc := val}_{u}$$

where φ is Java Card DL formula, loc a program variable, attribute or array access expression and val a term. The quantification and condition part are optional.

Two updates u_1, u_2 of the above form can be composed in parallel

$$u_1 \texttt{||} u_2$$

Application of an update on a formula or term results again in a formula resp. term (see formula/term definition).

Programs

An instance of the logic Java Card DL is always defined wrt. a context program declaring all classes and interfaces.

Programs used in Java Card DL formulas are actually lists of Java Card statements that are treated exactly as if inside a static method of a class in the default package.

Java Card DL extends Java Card *only* by two additional statements:

The Method-Frame statement surrounds a method body when it is inlined during a method invocation. The method-frame captures information like the current scope (`source`) and optionally, if not static, the receiver (`this`) of the method call and, if not void, the variable that is assigned the return value:

```
method-frame( result->program variable,
              source=classname,
              this=reference):{
          statement list
}
```

The Method Body statement is a placeholder for an actual method body implementation. For example, dynamic dispatching a method results in an **`if`** cascade and instead of immediately inlining the different method bodies in each branch the method body statement is used. Its syntax is:

$$resultVar\texttt{=}receiver.m\texttt{(}arg_1\texttt{,}\ldots\texttt{,}arg_n\texttt{)@}T$$

where T denotes the type of the concrete method *implementation.*

Contracts and Invariants

Contracts

A contract $\mathcal{C}_m := (pre, post, mod, term.\ marker)$ for a method m consists of

- a Java Card DL formula *pre* expressing the contracts precondition,
- a Java Card DL formula *post* expressing the contracts postcondition,
- a set of locations *mod* that might be changed and
- a termination marker *term. marker* indicating if the contract asserts termination or not.

Contracts are usually specified in JML or OCL, but can also be expressed in Java Card DL with a KeY problem file description (short: *dotkey* file).

— KeY —

```
\contracts {
 uniqueContractName {
   \programVariables {
      ResultType result;
      ReceiverType receiver;
      FirstArgType arg1;
      ...
   }

   pre ->
    \<{
       result = receiver.m(arg1,...,argN)@T
    }\> post
   \modifies { locations }
   \displayname "user − friendly name"
 }
}
```

— KeY —

If the contract should not guarantee termination, use box modality instead of diamond modality.

The pre-state value of an attribute a declared in type T can be accessed in postconditions via T`::`a`@pre(`o`)` resp. $ar[idx]$`@pre` if the prestate value of ar at index idx is accessed. *Attention*: `@pre` does not cause evaluation of a, ar or idx in the pre-state.

If a method throws an exception, it is possible to specify also the exceptional case in a contract:

— KeY —

```
\contracts {
 uniqueContractName {
   \programVariables { ... }
   pre ->
   \<{
       #catchAll(java.lang.Throwable exc) {
         result = receiver.m(arg1,...,argN)@T
       }
   }\> (
          (exc = null -> post_normal) &
          (exc != null -> post_exceptional)
       )
   \modifies { locations }
   \displayname "user − friendly name"
 }
}
```

— KeY —

Invariants

An invariant can be expressed in a similar way like contracts in KeY problem files.

— KeY —

```
\invariants(pkg.Class1 self) {
    invariant1 {
       self.attribute != null  & ...

      \displayname "My␣first␣invariant"
    };

    invariant2 {
      self.attribute2 != null & ...
    };
}
```

— KeY —

The invariant section must be declared after the `\contracts` section, if one exists. There can be arbitrary many invariant sections.