VDMJ High Precision Guide

The standard build of VDMJ performs arithmetic using Java longs (64-bits) for integer types and Java doubles for real types. This is sufficient for most model simulations, but there is an alternative build of VDMJ available that uses *BigIntegers* and *BigDecimals* to represent types. The alternative build is called the *high precision* build, and is available in the jar file called *vdmj-n.n.n-P.jar*.

The high precision build is exactly the same as the standard build for most purposes, and models that do not require high precision arithmetic will function in the same way, unless they depend on floating point results. Calculations are a few percent slower than the standard build for comparable precision, becoming progressively slower as the precision is increased.

The high precision build adds a new option to the command line, *-precision*, to specify the floating point precision required. By default this is 100, meaning floating point results are calculated to 100 significant figures. For example:

```
$ java -jar vdmj-4.0.0-P.jar -vdmsl -i stdlib/MATH.vdm -precision 50
Parsed 1 module in 0.292 secs. No syntax errors
Type checked 1 module in 0.015 secs. No type errors
Initialized 1 module in 0.238 secs.
Interpreter started
> precision
Decimal precision = 50
> print exp(1)
= 2.71828182845904523536028747135266249775724709369995957496696762772407663
Executed in 0.063 secs.

> precision 100
Decimal precision = 100
> print exp(1)
= 2.7182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274274663919320030599218174135966290435729003342952605956307381323286279434907632338298807531952510190115738341879307021540891499348841675092447614606680822648001684774118537423454424371075390777449920695517027618386062613313845830007520449338265602976067371132007093287091274437470472306969772093101416928368190255151086574637721112523897844250569536967707854499699679468644549059879316368892300987931277361782154249992295763514822082698951936680331825288693984964651058209392398294887933203625094431173012381970684161403970198376793206832823764648042953118023287825098194558153017567173613320698112509961818815930416903515988851934580727386673858942287922849989208680582574927961048419844436346324496848756023362482704197862320900216099023530436994184914631409343173814364054625315209618369088707016768396424378140592714563549061303107208510383750510115747704171898610687396965521267154688957035035
Executed in 0.919 secs.
>
```

This example shows the high precision jar being started with the MATH library and an initial precision of 50. The *precision* command line option displays and/or changes the setting, so the three examples show the evaluation of the Euler constant, *e*, to 50, 100 and 1000 decimal places.

Integer calculations are not limited by the precision option, and are always produced to full accuracy, though ultimately the Java *BigInteger* type is limited in the number of digits it can hold. For example:

```
> precision 50
Decimal precision = 50
```

```
> print 123 ** 456
= 992500687720988567008314620574696326372959408198869005198162988813828671047493990779211286614261446380554242369362718724928003527416499021181438196726015699981001207904967595176364654458956257416098662099005001984071532446047789680169630280503102614176159144687299182406854878786176459769390634643579861657117309763994785076492286863414669671679101266533421349427448514638999274870924866109771461127635671016726459531321964814393398730170881404146612711985003332557130961423351514146306516830655187840812036784877030028020820912366035190262568806244996817813872275740354848312715156831237421490955692604636096559777009388445806119312464951662086955403136981400116380273225662526897808381363518287953142721621112222311709017156123557013475523715300136938553798348656670600146433024591004297836539669137830022907842834556282833554705299329560514844771293338811599302127586876027950885792304316616960102321873904366016141456032419023866634425201607355665 61
Executed in 0.01 secs.
>
```

Here we see that the decimal precision is set to 50, but the value of 123 raised to the power 456 has all of its digits returned, rather than the first 50 significant digits and the rest zeros (which would happen for a floating point calculation).

Although high precision values can be used anywhere in a VDM specification, the following uses will be limited to the least significant 64 bits:

- The field number in an expression like, *record.#3*
- The number of iterations in a map or function expression, like x *** 3*.
- The index of a sequence.
- The arguments passed to *cycles*, *duration*, *periodic*, and *sporadic* statements.
- The *setPriority* argument for a CPU object in VDM-RT
- The number of repeats in a *traces* expression, like *op(){3}*.

Considering that a 64 bit value can be up to about 10^{18} there is no significant disadvantage in these limitations.

Note that if you use the VDMJUnit test framework, you need to link with the corresponding high precision version of the jar, called *vdmjunit-n.n.n-P.jar*. The difference is that the *runInt* and *runReal* methods in *VDMJUnitTest* return *BigInteger* and *BigDecimal* objects respectively, rather than returning *int* and *double*.