CS412 Lab session 1

An introduction to the B toolkit

Some preliminaries

A Community Edition of the Atelier B toolkit is installed on the Linux machines in the department. You can also download it on your own machine from: `https://www.atelierb.eu/en/download/`
On a Windows machine you need to run as Admin. The administrator manual has further info on set up procedures.

Starting the toolkit on department machines

Atelier B on the lab machines is run (from a command line) by typing: `startAB` which opens an Atelier B window. The department may not be running the latest version but basic functionality should be the same.

There are a number of help manuals which are all available here: `/package/Atelier_B/documentation/manual`. They're also linked on the module webpage. Note that in the department the help link inside the B tool won't work (due to an incompatibility between QT libraries).

Possible issues

Don't use spaces in any filenames you use.

You may find you need to use full pathnames for files (although you shouldn't have to enter filenames often).

Don't try to cut and paste from a pdf into the B editor. It may look ok initially but can cause invisible characters to become embedded. The only solution then is to start a new abstract machine.

This point should not affect new users of the tool, but if you get an error report concerning "bbatch" try deleting all existing workspaces, exiting and restarting.

Writing a specification

To start this week's lab session, create a new workspace by clicking on the yellow button with a plus sign (located towards the top left of the screen). Check the "software development" option and enter a "Project name", eg: `lab1`. You'll get the option to select the path for the directories it creates. The workspace section (to left of screen) should now include `lab1` and if you click it, it will expand to show a list of different areas: `Components`, `Definitions`, etc. Suppose we wish to create a component for the example1 machine (which is given at the back of this lab sheet). Click on the blue button with a plus sign (4th in from the left) to add a new component. Type in the name of the component (`example1`) and tell it the type (`Machine`) completing by clicking on `Finish`.
A template `example1` machine is created and should be listed in the Components for this project. A large coloured box with the component name is also created. (If you want to change this display to a less alarming text list, click on "classical view" just above the display.) Double click on the name of your component to open up an editor window and you can type in the machine details. You're not forced to use this editor (and you may get annoyed with the way it imposes structure) but it does conveniently display a list of all B symbols and the machine-readable notation for them. The `View` button at the top allows you to select what windows you want visible. Let's assume for now that we're working with the B editor.
Type in the `example1` machine exactly as it appears below - yes, there are some mistakes! Keywords will automatically appear as bold blue type. Obviously, save as you go along!

Analysing a specification

When you save a file, obvious syntactic errors will be reported in the Outline hand section of the edit window. If the Outline is not showing by default, click on the View tab in the Edit window to select it. If you've entered some wierd and wonderful errors of your own you may see different things, but for the specification as stated you'd see a list of error reports with the first one referring to a parse error. The token it refers to is also underlined in the edit window. What's required here is obviously a set subtraction so perhaps we've used the wrong symbol. The obvious thing is to check in the list of B symbols (if this isn't already open you can click on View then check the B symbols option). Now, I didn't do this on purpose, but it seems that there

is no entry for subtraction under the set section! We could probably have a good guess that it's the same as arithmetic subtraction - or else go back and check the notes.
Try correcting the error and save again. All the previous messages should disappear and this time we get a message about the identifier `maxval` in the `enter` operation. Hopefully you can quickly spot and correct the problem with this. There are now no error reports in the overview window.
The specification is now syntactically correct. In fact, this stage of the analysis can pick up a number of minor type errors too. But to do a full type check, go back to the main Atelier B window, right-click on the `example1` component and select the Type Check option. There are no type errors in this specification, and an "OK" mark is added to the component.
At times, you may see errors reported in the component's outline window which are marked as: (B0Check). Do not worry about any of these for now. They are giving a warning and we'll talk about what they mean later.

Things to try

Try inserting a variety of deliberate mistakes into the specification. Take note of the different error messages they produce.
Add the following operations. Remember that you can look up symbols in the B Symbols window of the B editor. There's also a link to a B syntax sheet on the module webpage. Add operations:

- to find the minimum number in the set;
- to tell you the number of elements in the set;
- to find whether a given number is in the set - output 'yes' if it is, 'no' otherwise. (NB: for a conditional statement you can use IF ... THEN ... ELSE.)
- Think of another operation you might provide and write it.

You can add comments by enclosing them in $/*\ldots*/$.
As you alter the machine, analyse it (save, correct syntax, type check) as before. It's generally a good idea to analyse as you go along (eg: after each operation is added) otherwise, you may end up with a whole load of confusing cascading errors.

A second example

The `example1` machine was given in machine-readable format. You may well have to translate from a "pretty" document to this form. Look at the `entrysys` machine from lecture 1 (also given below) and the suggested operations for it. Create a new machine component as you did before, this time for `entrysys`. Enter and analyse your machine, correcting any mistakes. Work through the questions given below the specification.

More of a challenge

Go back to the `example1` machine. Add an operation to output the average value of all the elements in the set. You'll probably find this one a bit harder because the syntax is a little fiddly. The list of B symbols may give you a good idea of what the appropriate operator will be. To find out more, try going to the B language reference manual (available on our website). The Arithmetical Expressions sections would seem to be a good place to look! Try to work out what you should use and how to write the operation. Have a go by yourself first. If you have trouble - ask your seminar tutor for help.

Animating a specification

So far you've been working with the tool and getting feedback from syntax and type checking. This doesn't tell you whether what you've written is doing what you really want or whether you've written a verifiably correct specification. The latter requires proof and we'll get to that another time. The former can be assisted by "animating" the specification. Remember, this is not a program so you can't expect to run it. However, an animator can help you explore behaviour within certain limits. The tool we'll use is ProB.

Instructions for using ProB

ProB is a tool that can be used to animate B specifications and although you won't be assessed on using this tool it will be helpful throughout the module for basic "sanity checking" of your specifications. ProB can be run stand-alone (in which case you just tell it which specification file you want it to open) or it can be linked to AtelierB so that specifications can be animated from within AtelierB.

If you're using a laptop you can download ProB for Windows, Linux and OSX from `http://www.stups.uni-duesseldorf.de/ProB/`. To link it to AtelierB, go into ProB's Help section and select "Install AtelierB 4 plugin". You can then input the directory where AtelierB is located.

Using ProB

In the lab you can run ProB stand-alone by typing `startprob`. Open the file you want to animate. Let's look at one of the specifications we've just written, say example1. The specification appears in the top window. The 3 windows below show:
1) the state of the machine, that is, the current values of its variables;
2) the operations whose preconditions are true in the current state;
3) the history of operations leading to the state you've reached.
The state window initially shows that it has picked limits for the set of numbers it will work with as it can only show finite behaviour. We'll see another time how to alter the values it uses for things like MAXINT.
The only operation offered initially is SETUP_CONSTANTS and unsurprisingly it offers to set the constant to the value we stated (but when not specified you need to make a choice to set values up). Double click on this and it'll set the value in the state window.
Next, your only option is initialisation. Click on this to view the effect. Depending on how you wrote your operations, you may get warnings in red in the state window - check them out if you do. Do you need to alter your specification?
Now all the enabled operations are shown. Why does it only allow you to enter numbers up to a certain value? Try clicking on a succession of operations and viewing the outcome. You can undo and redo steps by clicking the blue left/right arrows.
The enabled operations show you the outputs that will arise from each operation according to the inputs provided.
ProB does a lot more than this, but even at this level it's a great help in testing what you've written and debugging if needed. Feel free to explore the animator and learn more from its documentation. Remember that what it shows you is a guide only: it is not guaranteeing correctness and can only show very limited finite behaviour. We'll look at a few more useful things it can do next time.

Try this ...

Reset the animation (click Animate then Reset). Run the set up and initialisation operations. Now run the enter(1) operation followed by the enter(1) operation again. What does the state look like? Check you understand why.

When you've finished your work

Make sure you've saved any recent changes and exit the tool.
If you don't have time to finish the examples on this sheet today, try to work through them before the next lab session. If you've worked through these ones quickly, you could try experimenting with other machines from lectures or from the book - or that you invent yourself.

A first example

```
MACHINE           example1

CONSTANTS         max_val

PROPERTIES        max_val = 99999

VARIABLES         numset

INVARIANT         numset <: NAT1

INITIALISATION    numset := {}

OPERATIONS

  enter(new) =
    PRE
      new : NAT1 & new <= maxval
    THEN
      numset := numset \/ {new}
    END;

  mx  <-- maximum =
    PRE
      numset /= {}
    THEN
      mx := max(numset)
    END;

  remove(new) =
    PRE
      new : NAT
    THEN
      numset := numset \ {new}
    END

END
```

A second example

MACHINE *entrysys*
SETS *PID*
VARIABLES $inside, maxin$
INVARIANT $inside \subseteq PID \ \wedge$
$maxin \in \mathbb{N}_1 \ \wedge$
$card(inside) \leq maxin$
INITIALISATION
$inside := \{\} \ || \ maxin := 500$
OPERATIONS
$enter(pp) \mathrel{\widehat{=}}$
PRE
$pp \in PID \ \wedge \ pp \notin inside \ \wedge$
$card(inside) < maxin$
THEN
$inside := inside \cup \{pp\}$
END
END

- Write an `exit` operation for a person leaving.
- Write an operation, `empty`, which reflects the building being emptied all in one go (for a fire alarm perhaps?)
- Write an operation, `check`, which returns 1 if the person input is in the building and 0 if not.
- Rewrite the specification so that the `check` operation outputs "yes" if the person is found and "no" otherwise.
- Rewrite the specification so that the `enter` operation outputs the number of spaces remaining after this person has been added.
- Rewrite the specification so that `maxin` is represented as a constant rather than a variable.
- Add the variable `banned` to the specification to represent all people banned from entering the building. Make sure the invariant is updated to reflect the intended nature of this component. Check all the current operations and change as appropriate.
- Add an operation `ban` to ban a person.
- What does your version of `ban` do in the case that the person to be banned is in the building? Alter it so that it ejects the person as well as banning them if it doesn't do so already.