GCD Static Checking Example

Abstract

This example shows how to use contracts to prove some arithmetic properties of the greatest common denominator computation.

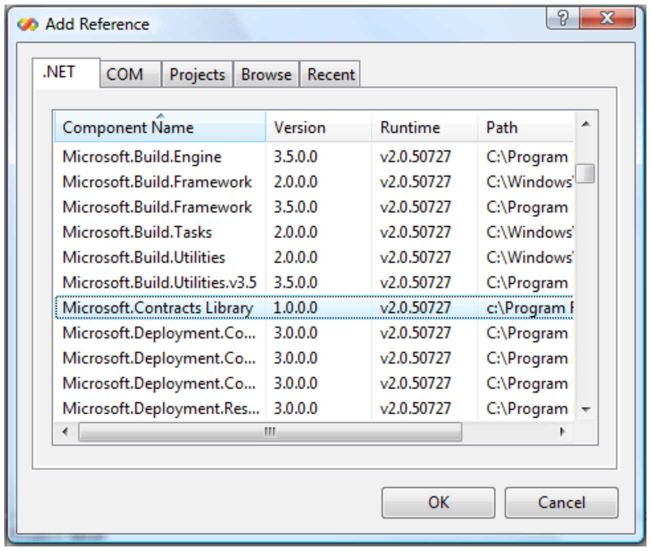
1 Adding the Contract Library Reference

If you are using Visual Studio 2008, or if you for some reason want to target a pre-v4 .NET runtime, then you need to:

- Change the target framework of the project.
- Manually add a reference to Microsoft.Contracts.dll

Otherwise, you may skip this section and go directly the next section!

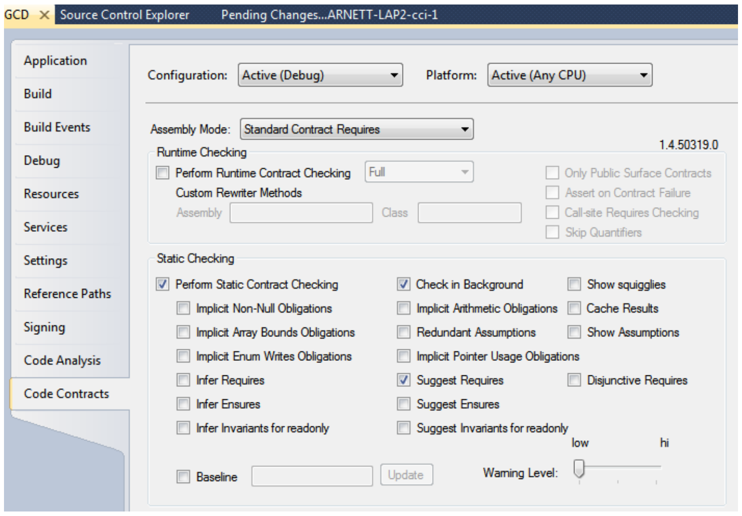
To add the reference, open the BinarySearch solution and right-click on References in the BinarySearch project and select Add Reference. Find the Microsoft.Contracts library in the .NET tab as shown below and click OK.



Component Name	Version	Runtime	Path
Microsoft.Build.Engine	3.5.0.0	v2.0.50727	C:\Program
Microsoft.Build.Framework	2.0.0.0	v2.0.50727	C:\Windows
Microsoft.Build.Framework	3.5.0.0	v2.0.50727	C:\Program
Microsoft.Build.Tasks	2.0.0.0	v2.0.50727	C:\Windows
Microsoft.Build.Utilities	2.0.0.0	v2.0.50727	C:\Windows
Microsoft.Build.Utilities.v3.5	3.5.0.0	v2.0.50727	C:\Program
Microsoft.Contracts Library	1.0.0.0	v2.0.50727	c:\Program
Microsoft.Deployment.Co...	3.0.0.0	v2.0.50727	C:\Program
Microsoft.Deployment.Co...	3.0.0.0	v2.0.50727	C:\Program
Microsoft.Deployment.Co...	3.0.0.0	v2.0.50727	C:\Program
Microsoft.Deployment.Res...	3.0.0.0	v2.0.50727	C:\Program

2 Sample Walkthrough

After adding the proper reference, go to the Properties of project GCD, select the Code Contracts pane (at the bottom), and enable static checking by clicking on the checkbox as shown in this screenshot:



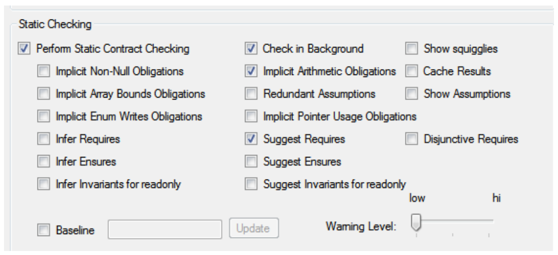
Configuration: Active (Debug) Platform: Active (Any CPU)

Assembly Mode: Standard Contract Requires 1.4.50319.0

Runtime Checking
- [ ] Perform Runtime Contract Checking Full
- Custom Rewriter Methods Assembly Class
- [ ] Only Public Surface Contracts
- [ ] Assert on Contract Failure
- [ ] Call-site Requires Checking
- [ ] Skip Quantifiers

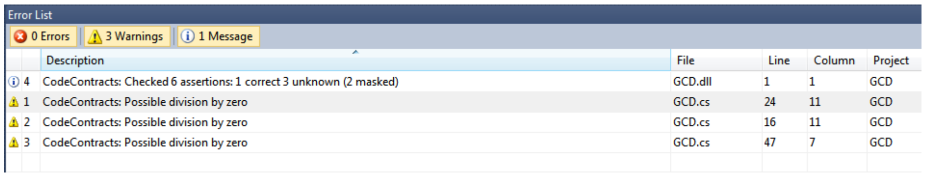
Static Checking
- [✓] Perform Static Contract Checking
- [ ] Implicit Non-Null Obligations
- [ ] Implicit Array Bounds Obligations
- [ ] Implicit Enum Writes Obligations
- [ ] Infer Requires
- [ ] Infer Ensures
- [ ] Infer Invariants for readonly
- [✓] Check in Background
- [ ] Implicit Arithmetic Obligations
- [ ] Redundant Assumptions
- [ ] Implicit Pointer Usage Obligations
- [✓] Suggest Requires
- [ ] Suggest Ensures
- [ ] Suggest Invariants for readonly
- [ ] Show squigglies
- [ ] Cache Results
- [ ] Show Assumptions
- [ ] Disjunctive Requires
- [ ] Baseline Update
- Warning Level: low – hi

Then build the example. There should be no warnings or errors at this point. To get static checking of arithmetic properties, such as division by zero, we need to enable that explicitly by checking the “Implicit Arithmetic Obligations” checkbox as shown in the following screen shot:



Static Checking
- [✓] Perform Static Contract Checking
- [ ] Implicit Non-Null Obligations
- [ ] Implicit Array Bounds Obligations
- [ ] Implicit Enum Writes Obligations
- [ ] Infer Requires
- [ ] Infer Ensures
- [ ] Infer Invariants for readonly
- [✓] Check in Background
- [✓] Implicit Arithmetic Obligations
- [ ] Redundant Assumptions
- [ ] Implicit Pointer Usage Obligations
- [✓] Suggest Requires
- [ ] Suggest Ensures
- [ ] Suggest Invariants for readonly
- [ ] Show squigglies
- [ ] Cache Results
- [ ] Show Assumptions
- [ ] Disjunctive Requires
- [ ] Baseline Update
- Warning Level: low – hi

Go ahead and add this option, then build again. The warning list should now display three warnings about possible division by zero:



Error List

0 Errors | 3 Warnings | 1 Message

	Description	File	Line	Column	Project
4	CodeContracts: Checked 6 assertions: 1 correct 3 unknown (2 masked)	GCD.dll	1	1	GCD
1	CodeContracts: Possible division by zero	GCD.cs	24	11	GCD
2	CodeContracts: Possible division by zero	GCD.cs	16	11	GCD
3	CodeContracts: Possible division by zero	GCD.cs	47	7	GCD

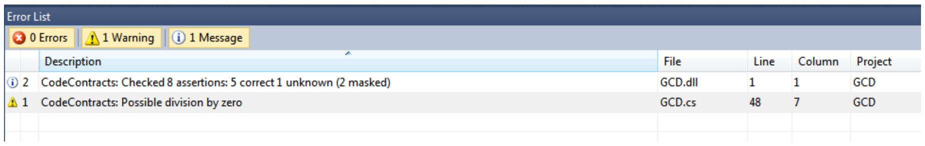
Let’s try to write some contracts to make sure we won’t run into these division by zero problems. Double click on the first warning. To avoid the division by zero of the code `x %= y`, we can add the following precondition to method GCD

```
  Contract.Requires(y > 0);
```

The second warning is about the similar division by x, so we add a similar precondition for it:

```
  Contract.Requires(x > 0);
```

Add those two preconditions and build again. You should now get the following warnings:



Error List

0 Errors | 1 Warning | 1 Message

	Description	File	Line	Column	Project
2	CodeContracts: Checked 8 assertions: 5 correct 1 unknown (2 masked)	GCD.dll	1	1	GCD
1	CodeContracts: Possible division by zero	GCD.cs	48	7	GCD

The possible division by zero remaining is in the NormalizedRational method, when dividing by the gcd value. The GCD should never be zero, and in fact due to our preconditions on the GCD method, our GCD will always be positive. So let's write a postcondition on GCD that makes this explicit. The contracts on GCD should now look as follows:

```
public static int GCD(int x, int y)
{
  Contract.Requires(x > 0);
  Contract.Requires(y > 0);
  Contract.Ensures(Contract.Result<int>() > 0);
```

Write the **Ensures** and rebuild. The checker should issue no more warnings.