MAT 7003 : Mathematical Foundations
(for Software Engineering)

J Paul Gibson, A207

paul.gibson@telecom-sudparis.eu

http://www-public.telecom-sudparis.eu/~gibson/Teaching/CSC7003/

Rigour - Answers

/~gibson/Teaching/CSC7003/L10-RigourAnswers.pdf
QUESTION: A TRS for formally defining if a number is prime

**Note:** easier to do in other formal languages/methods because the necessary concepts (like integers and lists are part of the language)

But, with the TRS we define just what we need and use it only where needed.

In software process it is this targetting (with the minimum force necessary) which is best …

**Question:** can you write a TRS for deciding if a given number is prime?

**Hint:** if not, try to break the problem down into bits

For the lists model/properties we should (but don’t have to) move up a level of abstraction!

We introduce Abstract Data Types…. IMHO the most powerful and universally applicable software process formal methods tool.
POSSIBLE ANSWER:

A TRS for deciding if a number is composite

\[ x-ty-qz \rightarrow Cz \]

Add to the tq- system (for multiply):

**Proposed Rule:**

if \( Cx \) is not a theorem then \( Px \) is a theorem

**Question:** why may this not be acceptable for deciding if a number is prime?
A TRS for deciding if a number is prime

AXIOM

P--

xy DND x

REWRITE RULES

x DND y -> x DND xy

-- DNDz -> zDF--

zDFx and x-DNDz -> zDFx-

z-DFz -> Pz-

Question: Can you verify that this is correct?
**Question:** add remove operation

```plaintext
TYPE Set SORTS Int, Bool
OPNS
empty: -> Set
str: Set, int -> Set
add: Set, int -> Set
contains: Set, int -> Bool

EQNS for all s :Set, x,y:int
contains(empty, x) = false;
x eq y => contains(str(s,x), y) = true;
not (x eq y) =>
contains(str(s,x), y) = contains(s,y);
contains(s,x) => add(s,x) = s;
not(contains(s,x)) => add(s,x) = str(s,x)
ENDTYPE

remove: Set, int -> Set

remove (empty, x) = empty
x eq y =>
remove(str(s,x), y) = s;
not(x eq y) =>
remove(str(s,x), y) =
str(remove(s,y), x);
```

**Question:** add *union* operation

```plaintext
TYPE Set SORTS Int, Bool
OPNS
empty:-> Set
str: Set, int -> Set
add: Set, int -> Set
contains: Set, int -> Bool
EQNS for all s,s1,s2 :Set, x,y:int
contains(empty, x) = false;
x eq y => contains(str(s,x), y) = true;
not (x eq y) =>
contains(str(s,x), y) = contains(s,y);
contains(s,x) => add(s,x) = s;
not(contains(s,x)) => add(s,x) = str(s,x)
ENDTYPE

union: Set, Set -> Set

union (empty, s1) = s1;
union (str(s1,x), s2) =
union (s1, add(s2,x));
```
Question: add equality operation

```
TYPE Set SORTS Int, Bool
OPNS
empty:-> Set
str: Set, int -> Set
add: Set, int -> Set
contains: Set, int -> Bool
EQNS forall s,s1,s2 :Set, x,y:Int
contains(empty, x) = false;
x eq y => contains(str(s,x), y) = true;
not (x eq y) => contains(str(s,x), y) = contains(s,y);
contains(s,x) => add(s,x) = s;
not(contains(s,x)) => add(s,x) = str(s,x)
ENDTYPE

equals, subset:
Set, Set -> Bool

equals(s1, s2) =
subset(s1,s2) and
subset (s2,s1)

subset(empty, empty) = true;
subset(empty, str(s,x)) = true;
subset(str(s2,x), s1) =
contains (s1,x) and
subset(s2, s1);
```