CSC 7003 : Introduction to Software Engineering

J Paul Gibson, D311

paul.gibson@telecom-sudparis.eu

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

Rigour - Answers

http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/L6-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 targeting (with the minimum force necessary) which is best …

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

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

---

**Question:** Can you verify whether this is correct?

**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

```
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

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:

```
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);
```