Lecture 20: Div and Mod

Dave Naumann

Department of Computer Science
Stevens Institute of Technology

CS 135 Discrete Structures Spring 2015

Outline of lecture

Studying

What's difficult about Scheme? What does and doesn't work for you learning Scheme?

What's difficult about reading the Rosen text? What does and doesn't work for you? How do you cope with parts of the text that refer to stuff you haven't read, or don't remember?

Integers and number theory

Some applications:

- graphics (machines do integer arithmetic faster than real arithmetic —but there are issues in numerical approximation that we won't be covering)
- analysis of algorithms (e.g., counting the number of messages in a protocol for peer-to-peer file sharing)
- cryptography

And it's a relatively simple playground for practicing meticulously precise reasoning.

Division

Recall $b \mid c$ was defined to mean "b divides c", i.e., $\exists d\ (c = b * d)$.
Here and throughout this lecture, a, b, c, d range over integers.
Let the symbol $\mid$ bind less tightly than other arithmetic operators but more tightly than the propositional operators.

What if $b = 0$?

Properties with direct proofs:
$a \mid b \wedge a \mid c \rightarrow a \mid (b + c)$
$a \mid b \rightarrow \forall c\ (a \mid bc)$
$a \mid b \wedge b \mid c \rightarrow a \mid c$
$a \mid b \wedge a \mid c \rightarrow \forall m, n\ (a \mid (mb + nc))$
Review: what's a prime number?

Division

Recall $b \mid c$ was defined to mean "b divides c", i.e., $\exists d\ (c = b * d)$.
Here and throughout this lecture, a, b, c, d range over integers.
Let the symbol $\mid$ bind less tightly than other arithmetic operators but more tightly than the propositional operators.

What if $b = 0$?

Properties with direct proofs:
$a \mid b \wedge a \mid c \rightarrow a \mid (b + c)$
$a \mid b \rightarrow \forall c\ (a \mid bc)$
$a \mid b \wedge b \mid c \rightarrow a \mid c$
$a \mid b \wedge a \mid c \rightarrow \forall m, n\ (a \mid (mb + nc))$
Review: what's a prime number?

Division

Recall $b \mid c$ was defined to mean "b divides c", i.e., $\exists d\ (c = b * d)$.
Here and throughout this lecture, a, b, c, d range over integers.
Let the symbol $\mid$ bind less tightly than other arithmetic operators but more tightly than the propositional operators.

What if $b = 0$?

Properties with direct proofs:
$a \mid b \wedge a \mid c \rightarrow a \mid (b + c)$
$a \mid b \rightarrow \forall c\ (a \mid bc)$
$a \mid b \wedge b \mid c \rightarrow a \mid c$
$a \mid b \wedge a \mid c \rightarrow \forall m, n\ (a \mid (mb + nc))$
Review: what's a prime number?

Divides as a binary relation

Is $|$ symmetric? reflexive? transitive? antisymmetric?

What's the term for such a relation?

A relation R is *total* if $\forall x, y.\ (xRy \vee yRx)$.

Divides as a binary relation

Is $|$ symmetric? reflexive? transitive? antisymmetric?

What's the term for such a relation?

A relation R is *total* if $\forall x, y.\ (xRy \vee yRx)$.

Divides as a binary relation

Is $|$ symmetric? reflexive? transitive? antisymmetric?

What's the term for such a relation?

A relation R is *total* if $\forall x, y.\ (xRy \vee yRx)$.

Division, remainder

Theorem ("division algorithm"): for any a and any positive d, there are unique q and r such that $\boxed{a = dq + r \ \wedge \ 0 \leqslant r < d}$

Proof of existence, in case $a \geqslant 0$.
This algorithm maintains the invariants $a = dq + r$ and $0 \leqslant r$, and it establishes $r < d$.

```
q:= 0; r:= a;
while r ⩾ d {
   r:= r − d;
   q:= q + 1;
}
```

Note $-10 \ \mathbf{div}\ 3 = -4$ and $-10 \ \mathbf{mod}\ 3 = 2$

Notation: q is $a \ \mathbf{div}\ d$ and r is $a \ \mathbf{mod}\ d$

Scheme: (modulo a b), Python: a // b and a % b

Division, remainder

Theorem ("division algorithm"): for any a and any positive d, there are unique q and r such that $\boxed{a = dq + r \ \wedge \ 0 \leqslant r < d}$

Proof of existence, in case $a \geqslant 0$.
This algorithm maintains the invariants $a = dq + r$ and $0 \leqslant r$, and it establishes $r < d$.

```
q:= 0; r:= a;
while r ⩾ d {
   r:= r − d;
   q:= q + 1;
}
```

Note $-10 \operatorname{\mathbf{div}} 3 = -4$ and $-10 \operatorname{\mathbf{mod}} 3 = 2$

Notation: q is $a \operatorname{\mathbf{div}} d$ and r is $a \operatorname{\mathbf{mod}} d$

Scheme: (modulo a b), Python: a // b and a % b

Division, remainder

Theorem ("division algorithm"): for any a and any positive d, there are unique q and r such that $\boxed{a = dq + r \ \wedge \ 0 \leqslant r < d}$

Proof of existence, in case $a \geqslant 0$.
This algorithm maintains the invariants $a = dq + r$ and $0 \leqslant r$, and it establishes $r < d$.

```
q:= 0; r:= a;
while r ⩾ d {
   r:= r − d;
   q:= q + 1;
}
```

Note $-10 \ \mathbf{div} \ 3 = -4$ and $-10 \ \mathbf{mod} \ 3 = 2$

Notation: q is $a \ \mathbf{div} \ d$ and r is $a \ \mathbf{mod} \ d$

Scheme: (modulo a b), Python: a // b and a % b

Congruence modulo

Recall from lect19: for any set S, T and function $f : S \to T$, we can define equivalence relation E on S by $\boxed{xEy \equiv f(x) = f(y)}$

Definition 3 page 240 in Rosen:
for $m \in \mathbf{Z}^+$, "a is congruent to b modulo m" means $m \mid a - b$.

Notation: $a \equiv b (\mathrm{mod}\, m)$
(Alternate notation: $a \cong_m b$ emphasizes that $\cong_m$ is a binary relation.)

Theorem: $a \equiv b (\mathrm{mod}\, m)$ iff $a \bmod m = b \bmod m$

Congruence modulo

Recall from lect19: for any set S, T and function $f : S \to T$, we can define equivalence relation E on S by $\boxed{xEy \equiv f(x) = f(y)}$

Definition 3 page 240 in Rosen:
for $m \in \mathbf{Z}^+$, "a is congruent to b modulo m" means $m \mid a - b$.

Notation: $a \equiv b (\mathrm{mod}\ m)$
(Alternate notation: $a \cong_m b$ emphasizes that $\cong_m$ is a binary relation.)

Theorem: $a \equiv b (\mathrm{mod}\ m)$ iff $a \bmod m = b \bmod m$

Proof of theorem

Theorem: $a \equiv b (\mathrm{mod}\ m)$ iff $a \bmod m = b \bmod m$

Proof by mutual implication.

First we prove $a \bmod m = b \bmod m \rightarrow a \equiv b (\mathrm{mod}\ m)$

1. $a \bmod m = b \bmod m$ by assumption
2. $a = m(a \operatorname{div} m) + a \bmod m$ by def (division algo)
3. $b = m(b \operatorname{div} m) + b \bmod m$ by def
4. $a - b = m(a \operatorname{div} m) + a \bmod m - m(b \operatorname{div} m) - b \bmod m$
   from 2,3 by arith
5. $a - b = m(a \operatorname{div} m - b \operatorname{div} m) + a \bmod m - b \bmod m$
   from 4 by arith
6. $a - b = m(a \operatorname{div} m - b \operatorname{div} m)$ from 1,5 by arith
7. $m \mid a - b$ from 6 by def of $\mid$
8. $a \equiv b (\mathrm{mod}\ m)$ from 7 by def of $\equiv (\mathrm{mod}\ m)$

Now to prove $a \equiv b (\mathrm{mod}\ m) \rightarrow a \bmod m = b \bmod m$ ...

Congruence

Let R be the relation "in same room as" on people indoors. And the functions $wearing : people \to footgear$ and $location : people \to buildings$.

xRy implies $location(x) = location(y)$
but xRy doesn't imply $wearing(x) = wearing(y)$

We say $location$ respects the equivalence R.

Addition and multiplication respect $\cong_m$:
If $a \equiv b \pmod m$ and $c \equiv d \pmod m$ then

$a + c \equiv b + d \pmod m$ and $ac \equiv bd \pmod m$

But not necessarily $a^c \equiv b^d \pmod m$;
And $ac \equiv bc \pmod m$ doesn't imply $a \equiv b \pmod m$.

Congruence

Let R be the relation "in same room as" on people indoors. And the functions $wearing : people \to footgear$ and $location : people \to buildings$.

xRy implies $location(x) = location(y)$
but xRy doesn't imply $wearing(x) = wearing(y)$

We say $location$ respects the equivalence R.

Addition and multiplication respect $\cong_m$:
If $a \equiv b \pmod m$ and $c \equiv d \pmod m$ then

$a + c \equiv b + d \pmod m$ and $ac \equiv bd \pmod m$

But not necessarily $a^c \equiv b^d \pmod m$;
And $ac \equiv bc \pmod m$ doesn't imply $a \equiv b \pmod m$.

Congruence

Let R be the relation "in same room as" on people indoors. And the functions $wearing : people \to footgear$ and $location : people \to buildings$.

xRy implies $location(x) = location(y)$
but xRy doesn't imply $wearing(x) = wearing(y)$

We say $location$ respects the equivalence R.

Addition and multiplication respect $\cong_m$:
If $a \equiv b \pmod m$ and $c \equiv d \pmod m$ then

$a + c \equiv b + d \pmod m$ and $ac \equiv bd \pmod m$

But not necessarily $a^c \equiv b^d \pmod m$;
And $ac \equiv bc \pmod m$ doesn't imply $a \equiv b \pmod m$.

Congruence

Let R be the relation "in same room as" on people indoors. And the functions $wearing : people \to footgear$ and $location : people \to buildings$.

xRy implies $location(x) = location(y)$
but xRy doesn't imply $wearing(x) = wearing(y)$

We say $location$ respects the equivalence R.

Addition and multiplication respect $\cong_m$:
If $a \equiv b \pmod m$ and $c \equiv d \pmod m$ then

$a + c \equiv b + d \pmod m$ and $ac \equiv bd \pmod m$

But not necessarily $a^c \equiv b^d \pmod m$;
And $ac \equiv bc \pmod m$ doesn't imply $a \equiv b \pmod m$.

Applications of modular arithmetic

Hash function to map data to index in size-N array:
$h(x) = dataToInt(x) \bmod N$

Caesar's cipher maps "zap" to "cds". Number the letters A..Z as 0..25, define $encrypt(x) = (x + 3) \bmod 26$ and
$decrypt(x) = (x - 3) \bmod 26$

Does *encrypt* need to be injective? surjective?

Applications of modular arithmetic

Hash function to map data to index in size-N array:
$h(x) = dataToInt(x) \bmod N$

Caesar's cipher maps "zap" to "cds". Number the letters A..Z as 0..25, define $encrypt(x) = (x + 3) \bmod 26$ and
$decrypt(x) = (x - 3) \bmod 26$

Does $encrypt$ need to be injective? surjective?

Summary/exercises

Definition of "congruent modulo m", for $m > 0$, is

$$\boxed{a \equiv b (\bmod m) \quad \equiv \quad m \mid (a - b)}$$

div and **mod** are defined by this property: For $a \in \mathbf{Z}$ and $d \in \mathbf{Z}^+$, we have $a = d * (a \textbf{ div } d) + (a \textbf{ mod } d)$

$$(a + b) \textbf{ mod } m = ((a \textbf{ mod } m) + (b \textbf{ mod } m)) \textbf{ mod } m$$

$$ab \textbf{ mod } m = ((a \textbf{ mod } m)(b \textbf{ mod } m)) \textbf{ mod } m$$

Programming language

Scheme docs for operations on integer? values, allowing negative divident/divisor.

(remainder n m) Returns q with the same sign as n such that

- (abs q) is between 0 (inclusive) and (abs m) (exclusive), and
- (+ q (* m (quotient n m))) equals n

(modulo n m) Returns q with the same sign as m where

- (abs q) is between 0 (inclusive) and (abs m) (exclusive), and
- the difference between q and (- n (* m (quotient n m))) is a multiple of m.