CS 337 **Solution to Test 3** **5/15/09**

1. (Relational Databases)

   (a) $SL \bowtie IT \bowtie SIP$ appears in Table 1.

Store	Location	Item	Type	Price
Amazon	WA	Nikon Cool-Pix	Camera	240
Amazon	WA	Dell Inspiron	Computer	1200
Fry's	CA	Nikon Cool-Pix	Camera	250
Fry's	CA	Sony Cybershot	Camera	310
Fry's	TX	Nikon Cool-Pix	Camera	250
Fry's	TX	Sony Cybershot	Camera	310
Best Buy	TX	HP Laptop	Computer	1300
Best Buy	TX	Sony Cybershot	Camera	280
Olde Tire	TX	Firestone	Tire	500

Table 1: $SL \bowtie IT \bowtie SIP$: Stores, Locations, Items, Types, Prices

   (b) Define certain predicates.

   p is Type = Camera
   q is Price < 300
   r is Location = TX

   The stores in TX that sell a Camera for less than 300 is given by the following query.

$$\pi_{\text{Store}} \left(\sigma_{p \wedge q \wedge r}(SL \bowtie IT \bowtie SIP)\right)$$

   (c) The stores, locations, items and prices for all computers that are being sold, is given by the following query.

$$\pi_{\text{Store, Location, Item, Price}} \left(\sigma_{\text{Type = Computer}} (SL \bowtie IT \bowtie SIP)\right)$$

2. (Rabin-Karp String Matching)

   (a) I show the hash function values for every 4-bit string, in Table 2. Note that 1100 mod 3 = 12 mod 3 = 0, and 1100 mod 5 = 2.

0	1	1	1	0	0	1	1	0	1	0	0	0	1	1	0	0	1	1	1	input
			1	2	0	0	0	0	1	1	1	2	1	0	0	0	0	0	1	mod 3
			2	4	2	4	3	1	3	0	4	3	1	3	1	2	4	3	2	mod 5

Table 2: Rabin-Karp String Matching

   Successful matches are shown here with a bar over the string: $01\overline{1100}1101000\overline{1100}11101$.

(b) The computation of exclusive-or is very easy: given string axb, where a and b are bits and x is a bit string, and you have already computed m, the exclusive-or of ax, you can compute exclusive-or of xb as $m \oplus a \oplus b$. But it is a very bad idea to use exclusive-or, because the only possible hash values are 0 and 1; therefore, there will be a collision around half the time.

3. (KMP String Matching)

(a) Patterns whose prefixes have short cores are preferable because it lets you move more to the right in the text string in case of a failure in matching.

(b) We are given that the cores of all prefixes are the empty string. If the first symbol, a, occurs more than once in s, then there is a prefix axa, where x is some substring (possibly empty). This prefix has a non-empty core because a is below axa. So, we conclude that the first symbol does not occur anywhere else in s. Conversely, if the first symbol does not occur anywhere else in s, the core is empty for every prefix, from the definition of core.

(c) A shortest string whose core is "ababa" is "abababa". Suppose there is a shorter string with core "ababa"; then it has to be of the form "ababax", for some symbol "x". Since "ababa" is a core of "ababax", it is also a suffix; so, "x" = "a". But "ababa" is not a core of "ababaa".

4. (Parallel Recursion)

(a)

$$
\begin{array}{ll}
 & h\langle 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\rangle \\
= & \{\text{rewriting}\} \\
 & h(\langle 0\ 2\ 4\ 6\rangle\ \bowtie\ \langle 1\ 3\ 5\ 7\rangle) \\
= & \{h(p \bowtie q) = p \mid q\} \\
 & \langle 0\ 2\ 4\ 6\rangle\ \mid\ \langle 1\ 3\ 5\ 7\rangle \\
= & \{\text{rewriting}\} \\
 & \langle 0\ 2\ 4\ 6\ 1\ 3\ 5\ 7\rangle
\end{array}
$$

(b) The proof of $rev(rr(rev(rr\ u))) = u$ is by induction on the length of u. For $u = \langle x\rangle$,

$$
\begin{array}{ll}
 & rev(rr(rev(rr\ \langle x\rangle))) \\
= & \{\text{definition of } rr\} \\
 & rev(rr(rev\langle x\rangle)) \\
= & \{\text{definition of } rev\} \\
 & rev(rr\langle x\rangle) \\
= & \{\text{definition of } rr\} \\
 & rev\langle x\rangle \\
= & \{\text{definition of } rev\} \\
 & \langle x\rangle
\end{array}
$$

For $u = p \bowtie q$

$$
\begin{array}{cl}
 & rev(rr(rev(rr\ (p \bowtie q)))) \\
= & \{\text{definition of } rr\} \\
 & rev(rr(rev(q \bowtie (rr\ p)))) \\
= & \{\text{definition of } rev\} \\
 & rev(rr((rev(rr\ p)) \bowtie (rev\ q))) \\
= & \{\text{definition of } rr \text{ applied to } rr((rev(rr\ p)) \bowtie (rev\ q))\} \\
 & rev((rev\ q) \bowtie rr(rev(rr\ p))) \\
= & \{\text{definition of } rev\} \\
 & rev(rr(rev(rr\ p))) \bowtie (rev(rev\ q)) \\
= & \{\text{induction on } rr(rev(rr\ p))\} \\
 & p \bowtie (rev(rev\ q)) \\
= & \{rev(rev\ q) = q\} \\
 & p \bowtie q
\end{array}
$$

(c) In all cases, proof is by induction on i.

i. We show $u_{i+1} = u_i \bowtie v_i$, and $v_{i+1} = v_i \bowtie u_i$.
For $i = 0$, we have to show $u_1 = u_0 \bowtie v_0$, and $v_1 = v_0 \bowtie u_0$. Since u_0 and v_0 are singleton lists, $u_0 \bowtie v_0 = u_0 \mid v_0 = u_1$. The proof of $v_1 = v_0 \bowtie u_0$ is similar.
For $i > 0$,

$$
\begin{array}{cl}
 & u_{i+1} \\
= & \{\text{definition}\} \\
 & u_i \mid v_i \\
= & \{\text{induction; note that } i > 0\} \\
 & (u_{i-1} \bowtie v_{i-1}) \mid (v_{i-1} \bowtie u_{i-1}) \\
= & \{\text{commutativity law}\} \\
 & (u_{i-1} \mid v_{i-1}) \ \bowtie \ (v_{i-1} \mid u_{i-1}) \\
= & \{\text{definition; note that } i > 0\} \\
 & u_i \ \bowtie \ v_i
\end{array}
$$

The proof of $v_{i+1} = v_i \bowtie u_i$ is similar.

ii. We show u_i is the bit-wise complement of v_i. Write $\overline{v_i}$ for the complement of v_i.
For $i = 0$, $\overline{v_0} = \overline{\langle 1 \rangle} = \langle \overline{1} \rangle = \langle 0 \rangle = u_0$.
For $i + 1$,

$$
\begin{array}{cl}
 & \overline{v_{i+1}} \\
= & \{\text{definition of } v_{i+1}\} \\
 & \overline{v_i \mid u_i} \\
= & \{\text{distribute complementation}\} \\
 & \overline{v_i} \mid \overline{u_i} \\
= & \{\text{induction}\} \\
 & u_i \mid v_i \\
= & \{\text{definition of } u_{i+1}\} \\
 & u_{i+1}
\end{array}
$$

(d) See Figure below for data movement.

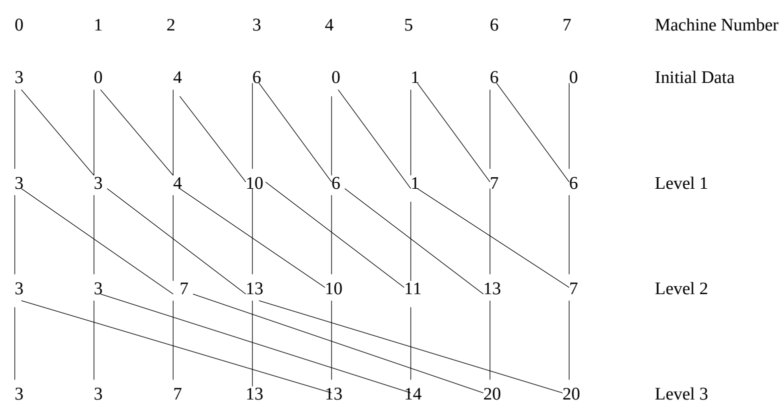

Figure 1: Prefix Sum