MFCS

Cartesian Products

Klaus Sutner

Carnegie Mellon University

Fall 2022

Unary Operations

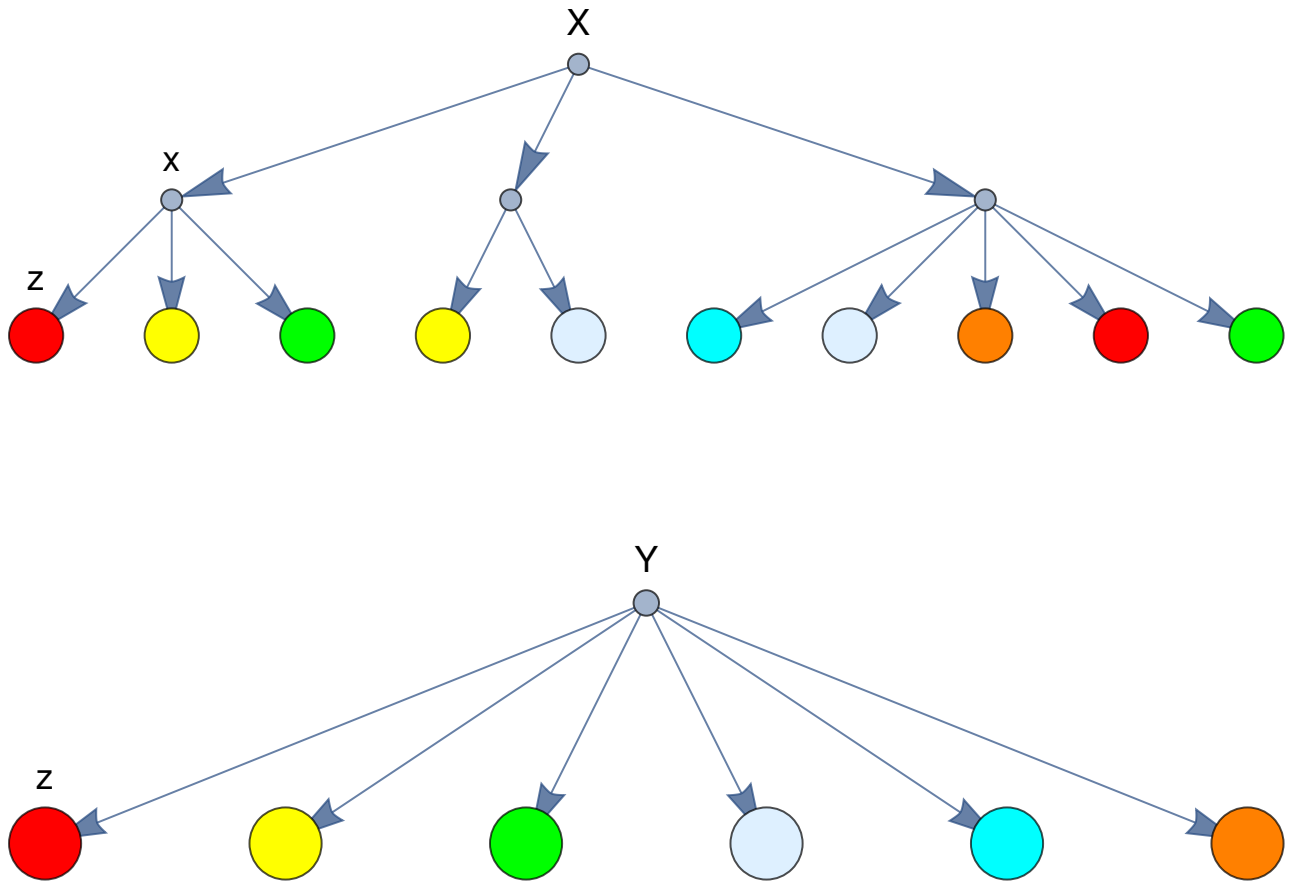
We have defined union and intersection as binary operators. That's fine, but it is actually more useful to introduce a unary version.

Definition

$$\bigcup X = \{\, z \mid \exists\, x\, (z \in x \wedge x \in X)\,\}$$

$$\bigcap X = \{\, z \mid \forall\, x\, (x \in X \Rightarrow z \in x)\,\}$$

To make sense of this, think of X as a family of sets, we want to union/intersect all of them.


X
x
z
Y
z

Rewrite

These definitions may be easier to read in the following form.

$$\bigcup X = \{\, z \mid \exists\, x \in X (z \in x) \,\}$$
$$\bigcap X = \{\, z \mid \forall\, x \in X\, (z \in x) \,\}$$

First note that nothing is lost, we can recover the binary version easily.

$$\bigcup \{a, b\} = a \cup b \qquad \bigcap \{a, b\} = a \cap b$$

Everything typechecks just fine: in our world, everything is a set, so we can ask about the elements of the elements of X.

Destroyer of Worlds

Repeated application of $\bigcup$ may destroy a set. Aka an excellent way to implement counters.

$$\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\}$$
$$\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$$
$$\{\emptyset, \{\emptyset\}\}$$
$$\{\emptyset\}$$
$$\emptyset$$

Exercise

Does this always work?
What would happen if we used intersection instead?

Warning

Is clear that

$$\bigcup \emptyset = \emptyset$$

But we need to be a bit careful with intersection:

$$\bigcap \emptyset = \text{universe of all sets}$$

So this is not a set and must be avoided.

For $\bigcap X$ we always must make sure that $X \neq \emptyset$.

Building Sets

Suppose we want the smallest set $A \subseteq \mathbb{N}$ such that

- $5 \in A$, and
- whenever $x \in A$, then $x + 13$ and $x + 29$ are also in A.

Here is a brutal way to build this set. Say $X \subseteq \mathbb{N}$ is *well-behaved* if $5 \in X$ and $x + 13$ and $x + 29$ in X whenever $x \in X$.

Done! We can simply set

$$A = \bigcap \{\, X \subseteq \mathbb{N} \mid X \text{ well-behaved} \,\}$$

Is this Legit?

Why does this definition work?

$$A = \bigcap \{ X \subseteq \mathbb{N} \mid X \text{ well-behaved} \}$$

First off, $X = \mathbb{N}$ is well-behaved, so it is one of the sets on the RHS. It follows that $A \subseteq \mathbb{N}$.

Second, if X_1 and X_2 are well-behaved, then $X_1 \cap X_2$ is also well-behaved. This also works for infinite intersections, so A is also well-behaved.

But by definition, A is the smallest well-behaved set. We're good.

Pretty Vague

One annoying feature of this construction is that we learn absolutely nothing about A, it exists in set-theory-lala-land and basta.

This type of reasoning is currently the accepted standard in math, though a small minority complains. In CS it is somewhat more disconcerting since we ultimately want to compute and this type of existence proof has no computational relevance.

We really would like to understand the layout of A, we would like to construct A in some reasonable way so we get more information. In this example, one can get a complete description of A with little effort.

Exercise

Explain A to a 10-year old.

Looks Weird

Another and deeper problem is that there is an annoying sense of circularity in this "construction."

> A is well-behaved, hence it is one of Xs on the RHS.
> So we are defining A in terms of itself.

Again, a few people protest against this circularity, but, for the most part, no one cares. This method is just too useful to be discarded.

Hilbert

From the paradise, that Cantor created for us, no-one shall be able to expel us.

D. Hilbert (1926)

Indexed Families

If you find unary operators scary, here is a way to avoid cognitive challenges: an indexed family of sets is a bunch of sets A_i where i ranges over some index set I (typically $[n]$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ or maybe some ordinal).

$$\bigcup_{i \in I} A_i = \{\, x \in A_i \mid i \in I \,\}$$

and likewise for $\bigcap_{i \in I} A_i$.

Nothing new here, this is just syntactic sugar for $\bigcup \{\, A_i \mid i \in I \,\}$. But sometimes it is helpful to have a clear enumeration of the elements of X.

This notation is particularly popular in calculus.

$$\bigcup_{\varepsilon>0} [0, 1-\varepsilon] = [0, 1) \subseteq \mathbb{R}$$

$$\bigcap_{\varepsilon>0} [0, 1+\varepsilon] = [0, 1] \subseteq \mathbb{R}$$

Exercise

As written, the index set is the positive reals. Change things so that the index set is $\mathbb{N}$, but the result is the same.

Cartesian Product

Suppose we have two sets and we want to form the set of all pairs with elements in the two sets. This will turn out to be one of the most useful constructions in set theory.

Definition

The Cartesian Product of two sets A and B is

$$A \times B = \{ (a, b) \mid a \in A, b \in B \}$$

For example, in geometry we can think of the plane as being the Cartesian product $\mathbb{R} \times \mathbb{R}$.

Note that these pairs are ordered, in general $(a, b) \neq (b, a)$.

Wait a minute . . .

So now we have messed up our little sandbox, we are dealing with

sets **element-of** **pairs**

Remember, we wanted to make do with sets and element-of only, we have just introduced a new type. Of course, everyone intuitively understands pairs, but there are pesky questions: what are the elements of a pair? Do they not have any? When are two pairs equal? And so on.

Equality of pairs is easy:

$$(a, b) = (c, d) \iff (a = c) \wedge (b = d)$$

But the other questions are messier.

Brilliant Idea

Maybe we can implement pairs as sets?

In other words, maybe we can define a binary operation π on sets that gives us pairs, but using only sets.

A little thought reveals that all we really need is

$$\pi(a, b) = \pi(c, d) \quad \text{implies} \quad (a = c) \wedge (b = d)$$

Of course, $\pi(a, b) = \{a, b\}$ fails miserably.

It's Not That Easy

Historically, it took some top-notch mathematicians several tries before they found a good way of doing this.

For example, here is an attempt by Norbert Wiener:

$$\pi(a, b) = \{\{\{a\}, \emptyset\}, \{\{b\}\}\}$$

Felix Hausdorff came up with a slightly better solution:

$$\pi(a, b) = \{\{a, \emptyset\}, \{b, \{\emptyset\}\}\}$$

Note we are essentially tagging a and b by 0 and 1.

Kuratowski to the Rescue

No one uses these, everyone relies on Kazimierz Kuratowski's definition.

Definition

The (Kuratowski) pair of x and y is

$$\pi(x, y) = \{\{x\}, \{x, y\}\}$$

Lemma

$\pi(u, v) = \pi(x, y)$ *implies* $u = x$ *and* $v = y$.

Exercise

Prove that this is really true and construct the actual unpairing functions (he unary operators may come in handy).

Keeping it Simple

From now on, whenever we see (a, b) we

- think of it intuitively as an ordered pair, but
- if necessary, we unfold using Kuratowski's definition.

We could write $\pi(u, v)$ everywhere, but that is just pedantic. Keeping notation simple is hugely important.

Bad notation clogs up your brain and slows you down, to the point of complete standstill.

Again: For humans it is critical **not**[†] to always unfold all definitions down to the most basic level. We understand what a pair is supposed to be, and we are happy to leave it at that.

Only under duress will we use Kuratowski's definition directly.

Think about programming in a high level language, say, C++. No one thinks about the corresponding assembler code while using all kinds of higher level machinery. Those who try go insane and get nothing done.

[†] In fact, this is also true for computers. Theorem provers based on set theory do not trace everything down to $\emptyset$.

A Property

Claim: $(A \times B) \cap (A \times C) = A \times (B \cap C)$.

Proof.

$x \in$ LHS implies $x \in A \times B$ and $x \in A \times C$. Hence $x = (a, b)$ and $x = (a', c)$ for some $a, a' \in A$, $b \in B$, $c \in C$.

By the property of pairs, $a = a'$ and $b = c$. But then $x = (a, b) \in$ RHS.

$x \in$ RHS means $x = (a, b)$ where $a \in A$ and $b \in B \cap C$. Hence $x \in A \cap B$ and $x \in A \cap C$. But then $x \in$ LHS.

□

Note that this is slightly more complicated than the proof of, say, distributivity last time. We need to reason about pairs in addition to purely "algebraic" properties.

Pairs can be extended to n-tuples for $n \geq 3$ by a recursive construction:

$$(a_1, \ldots, a_n) = ((a_1, \ldots, a_{n-1}), a_n)$$

Lemma

$(a_1, \ldots, a_n) = (b_1, \ldots, b_n)$ *implies* $a_i = b_i$.

n-tuples are essentially the same as finite sequences or lists, though the details of the definitions may differ. There is always a simple way to translate back and forth.

Longer Products

Using the same idea we can also generalize our Cartesian products:

$$\bigtimes_{i=1}^{n} A_i = \left(\bigtimes_{i=1}^{n-1} A_i\right) \times A_n$$

For example, ordinary Euclidean space could be modeled by $\bigtimes_{i=1}^{3} \mathbb{R}$, and n-dimensional space by $\bigtimes_{i=1}^{n} \mathbb{R}$.

Exercise (Left vs Right)

What if we used $(a_1, \ldots, a_k) = (a_1, (a_2, \ldots, a_k))$ *instead?*
Would this make a substantial difference?

Obnoxious Question: $n = 0, 1$???

Why would we care about this? Because arbitrary products in arithmetic are perfectly well defined, $\prod_{i=1}^{n} a_i$ makes sense for all $n \in \mathbb{N}$. We'd like to have a similar setup here.

$n > 2$: Use recursion.

$n = 2$: No problem, this is how we started.

$n = 1$: We could just use A_1, meaning a 1-tuple is just the thing itself[†].

$n = 0$: This is the hard part. In analogy to arithmetic products $\prod_{i=1}^{0} a_i = 1$ we can use any one-element set, say $\{\emptyset\}$.

[†] This is somewhat dicey, think about it.

Food for Thought

As one can see, our definition of arbitrary Cartesian products runs into a few obstructions. Nothing really serious, but bothersome.

Here is another approach. Define a k-tuple over A to be a map

$$t : [k] \to A$$

Basically a finite sequence or vector of elements of A.

Exercise

Develop Cartesian products based on these k-tuples.
Compare to our old Cartesian products.