فصل دوم:

اولویت ها: نات بعد اند بعد اور و ... .

اگر در درخت ما برگ هاش خالی باشد از introduction بهش تئوری میگوییم.

P یک گزاره هست وقتی براکت میگذاریم یعنی فرض میگیریم درست هست بالاش در توان شماره آن را میگذاریم تا موقع استفاده متوجه بشویم.

Introduction زیاد میکنه elimination کم میکنه.

در بالای خط کسری میگذاریم فرض میکنیم درست هست. داخل براکت اون فرض را discharge میکند. فرض کردیم p درست هست بعد به r رسیدیم یعنی یک درخت داشتیم یک جایی این فرض رو کردیم و یک جایی این نتیجه رو گرفتیم. بر اساس درستی p ما q را نتیجه گرفتیم.

Predicate همان preposition هست که راجب درستی یا غلط نمیتوانیم صحبت کنیم باید مشخص شود.

اسلاید پنجم:

Or را میتوانیم به تعلق و برعکس تبدیل کنیم. یا EXT-MEM. دو مجموعه مساوی هستند که هر عضو این عضو اون یکی باشد و برعکس و بین این ۲ AND هست.

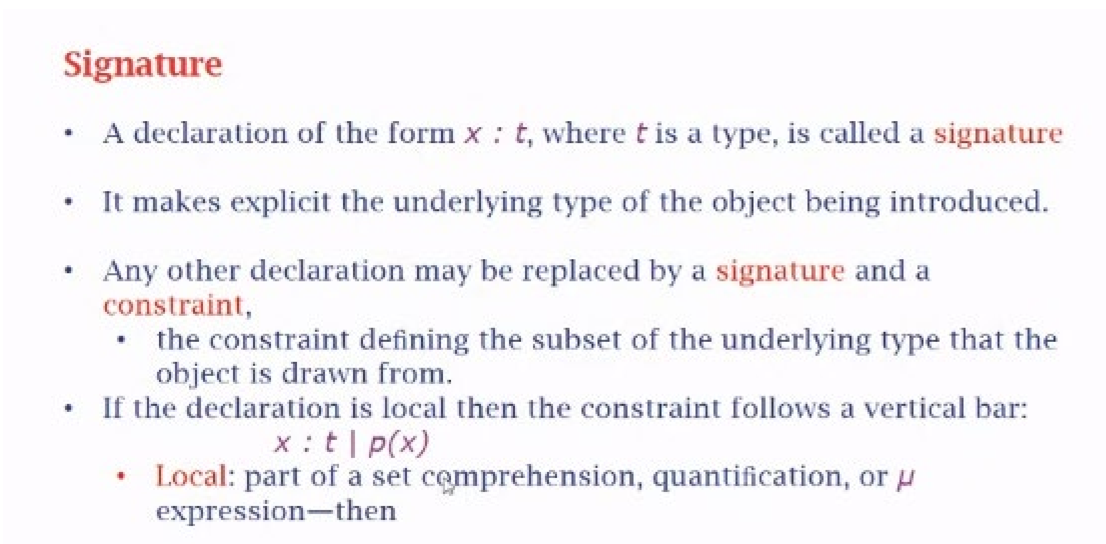
S زیر مجموعه اون هست که همه اعضاش داخل اون یکی باشد. تهی زیر مجموعه همه هست. در واقع وقتی | میزاریم یعنی یک زیر مجموعه از S که دارای خاصیت P هستند. بعد از E | اومد بعد تابع اومد ADDRESS(P) عناصر مجموعه دیگر پرسن نیستند بلکه آدرس هستند یا بولت بگذاریم. S متعلق به مجموعه توانی A یعنی یکی از زیر مجموعه های آن است. وقتی u اجتماع بزرگ پشت یک مجموعه میگذاری یعنی اون s مثلا خودش مجموعه ای از مجموعه ها هست و عناصر آن باید با هم اجتماع شوند. همین برای اشتراک. هر مجموعه ای میتواند تایپ باشد اما گاهی اوقات دیگر تایپ نیستند ساب ست تایپ دیگر هستند مثلا دانش اموز و معلم ساب ست پرسن هستند. اعضای اون یکی نتوانند عضو اون یکی باشند اگر باشند باید تایپ بزرگتر در نظر بگیری یعنی استادی دانشجو نباشد و برعکس اگر بود باید پرسن بگیری. ==S مجموعه تعریف میکنی. هر عنصری فقط متعلق باید به یک تایپ باشد. جز built it ها.

مجموعه ای شامل constant ها: color ::= red | orange| blue| green اینا متغیر نیستند دیگر.

این با این فرق میکند

Colors == {red, green, orange} بخاطر همین maximal هست چون بالایی ثابت هست عضوش کم و زیاد نمیشود خودمان گفتیم maximal این هست اما در مورد پایینی نه. اصلا مورد پایین را نمیشناسد درکی ندارد ازش دنبال متغیر جای آنها هست ولی بالایی اصلا متغیر نیست. در بالایی اعضا با همدیگر فرق میکنند ولی پایین متغیر هستند و میتوانند مقدار بگیرند و میتوانند با هم برابر باشند به بالایی free type هم گفته میشود. پس برای مثال دوم حتما باید اینها از قبل تعریف شده باشند و دوما لزومی هم ندارد از هم مجزا باشند چون متغیر هستند و همین مشکل هست چون آبی میتواند قرمز شود مثلا.

تایپ مجموعه تهی چیست؟ براکت ایکس میشود اون مجموعه که تهی جز توانی آن است.



X == e در واقع e نام دیگری برای X هست. X یک مجموعه میشود از این به بعد.

Consistency یعنی مقادیری پیدا بشود بعد از تعریف constant global یک مقدار ارضا پذیر هم پیدا شود یعنی یک متغیری پیدا شود there is باید باشد سور وجودی یعنی باید همراه اون سور وجودی هم بنویسیم.

اگر given type تهی باشد مشکل هست و ناسازگاری پیش میاد. میتواند ساب ست آن تهی باشد ولی خود تایپ نباید تهی باشد. Generic یعنی تایپ متفاوت داشته باشد، x[set of parameter] == e . تهی را میتوانی generic تعریف کنی چه عدد طبیعی تهی چه دانشجو تهی.

$$\mathbb{P}_1[T] == \{a : \mathbb{P}T \mid a \neq \emptyset\}$$

For the second generic symbol($\emptyset$), from the con
the empty set of elements from T

$$\mathbb{P}_1[\{0,1\}] == \{a : \mathbb{P}\{0,1\} \mid a \neq \emptyset\}$$
$$= \{\{0\}, \{1\}, \{0,1\}\}$$

مجموع همه زیر مجموعه ها بدون تهی. [x] تایپ های generic هست بالا لیست آنجا هست پس یک x از تایپ آن تعریف میشود بعد predicate ها.

These axiomatic definitions justify the following:

$$\frac{}{\emptyset[Car] \in \mathbb{P}\ Car \land \forall x : Car \cdot x \notin \emptyset[Car]}$$

Underscore یعنی دو تا پارامتر میگیرد.

Crowd : p person یعنی crowd زیر مجموعه person هست.

#s همون len(s) هست یا کاردینالیتی. P (p crowd) یعنی اینکه crowds مساوی همه زیر مجموعه، زیر مجموعه های person هست مجموع اونها هست. مجموعه ای از ساب ست ها میخواهیم پس دو تا پاور میخواهیم. یعنی تمام ست هایی که crowds هستند.

اسلاید هفتم:

Relation همون ضرب دکارتی یا زوج مرتب هست و x ⇔ y یعنی تمام relation بین این ۲ یا مجموعه توانی ضرب آنها که شامل مجموعه همه زوج مرتب ها میشود.

Example

The relation *drives* is used to record which makes of *Cars* are driven by the members of a small group of people *Drivers*.

$$Drivers == \{helen, indra, jim, kate\}$$

$$Cars == \{alfa, beetle, cortina, delorean\}$$

$$drives == \{helen \mapsto beetle,$$
$$indra \mapsto alfa,$$
$$jim \mapsto beetle,$$
$$kate \mapsto cortina\}$$

Then *drives* is an element of $Drivers \longleftrightarrow Cars$, and the statement *'Kate drives a cortina'* could be formalised as

$$kate \mapsto cortina \in drives.$$

در واقع drivers بین رابطه DRIVERS و CARS قرار دارد. از یک سری operator برای relation میتوانیم استفاده کنیم. مثلا میخواهیم بدانیم اشیا داخل این رابطه چی هست با dom R مشخص میکنیم یعنی از مجموعه مبدا چه کسانی شرکت کردند. دامنه میشود مبدا range میشود برد یا مقصد هایی که شرکت کردند.

محدودیت یا restriction برای دامنه سر مثلث به A اشاره میکند میگوییم عنصر اول یا مبدا باید داشته باشند و برای برد سر مثلث سمت B هست جا به جا شده و عنصر دوم رابطه باید اون شرط ها را داشته باشد.

مقصد و مبدا تایپ یکسان homogenous برعکس heterogenous. Id X میخواهیم بگیم هر عنصر x با خودش در ارتباط هست. reflexive از خاصیت homogenous هست که میگوید جدا از اینکه مبدا و مقصد یکسان بلکه هر عنصر با خودش باید رابطه داشته باشد. Reflexive [X] == . مثل رابطه کوچک تر مساوی که مثلا ۵ از ۵ هم کوچکتر هست هم مساوی.

Symmetry : اگر x به y هست باید y به x هم باشد منتها در R حتما یعنی عضو اون باشند. رابطه صحبت دو نفر با هم همین هست هم این با اون هم اون با این صحبت میکند.

Antisymmetric: میگوید حالا که این هست پس x , y یکسان هست.

Asymmetric: میگوید اگر x - y بود نباید y - x باشد.

If $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow Z$ then $R \mathbin{⨾} S$ denotes the relational composition of R and S.

$R \mathbin{⨾} S : X \leftrightarrow Z$ such that:

$$x \mapsto z \in R \mathbin{⨾} S \Leftrightarrow \exists y : Y \bullet x \mapsto y \in R \land y \mapsto z \in S$$

That is, two elements x and z are related by the composition $R \mathbin{⨾} S$ if there is an intermediate element y such that x is related to y and y is related to z.

الان buys یک رابطه جدید بین DRIVERS - FUELS هست. تولد هر ۳ تا هست هم reflexive یعنی هر عکس با تولد خودش در رابطه هست. symmetric هست یعنی جفت تولد هستند و تایپ یکی هستند یعنی تولد این هست تولد هم برای این هست. transitive هم هست.

Closure یعنی بر اساس یک ویژگی جدید بسته بشود. یعنی مثلا برای reflexive بزنی به خودش رابطه میگیرد. برای symmetric یعنی r u r^r اینکه اجتماع کنیم تا با هم مثل گفتگو کردن با هم در ارتباط باشند. رابطه را هعی میخواهیم با خودمان compose کنیم $r^1 == r$ و $r^2 == r:r$. برای transitive از همین استفاده میکنیم. به ازای هر direct یک مقصد برای همه همین رو در نظر بگیر هر بار در خودش ضرب کردی یکی بیشتر میشه.

اسلاید هشتم:

اگر قرار هست داخل رابطه از یک سمت دقیقا با یک ابجکت از سمت دیگر در ارتباط باشند تابع یا function نامیده میشود. partial میگوید برای همه مبدا ها تعیین نمیکنیم. نحوه تعریف تابع name_function: r1 - r2 بعد اگر روی بین دو تا r خط زدند یعنی partial هست.

تو تعریف تابع بالا تعریف میکنیم نوع تابع و رابطه را بعد پایین predicate میدهیم.

تابع را میخواهیم اپلای کنیم روی یک ارگومان باید چک کنیم که داخل دامنه هست یا نیست.

λ *declaration* | *constraint* $\bullet$ *result*

Example

$$\begin{array}{|l} double : \mathbb{N} \rightarrow \mathbb{N} \\ \hline double = \lambda\, m : \mathbb{N} \bullet m + m \end{array}$$

تابع میتواند injective باشد یک به یک. Surjective چند به چند. Bijective ترکیب این ۲ هست. ۱ به ۱ دقیقا هر کدام با یکی رابطه دارند البته اسرار نیست مجموعه مقصد کامل پوشیده شود در دومی میگیم مجموع مقصد همه باشد پوشیده باشند و در رابطه باشند.

FX میشود مجموعه محدود مجموعه. هر جا سور وجودی استفاده کردی بولت نیاز داری میتوانی محدود کردن یا دیوار | بگذاری یا نگذاری ولی بولت حتما نیاز هست. دو نقطه یعنی مقادیر بین ۲ تا عدد.

Hash همان len هست.

فصل نهم:

در sequence یک کالکشن میخواهیم هم ترتیب هم تکرار داشته باشد.

با یک برعکس میگیم یک ترتیبی از همین میخواهیم که مثلا فقط تو s I z باشند ترتیب هم مهم نیست همان ترتیبی که ظاهر شدند هستند.

Head , tail هم میتوانیم روی دنباله ها اعمال کنیم یکی به عنصر اول که فقط یک عنصر ولی tail یک مجموعه میدهد همه جز عنصر اولی. با همون شارپ هم طول دنباله ها را میتوانی بگیری.

Seq X میشود مجموعه تمام ست های محدود X. تمام عناصر این توالی از یک تایپ هستند یعنی وقتی اینطوری مینویسیم.

$$\langle a, b, c, b, e, d, f, b, g \rangle \upharpoonright \{a, c, d\} = \langle a, c, d \rangle$$

$$squash\ f = (\mu\, g : 1..\#f \rightarrowtail dom\ f \mid (g^{\sim} ; (\_+1) ; g) \subseteq (\_<\_)) ; f$$

زیر خط قرمز همون

شرط ascending order هست.

Bags : ترتیب مهم نیست فقط تعداد مهم هست. count: چند تا از یک عنصر در یک بگ هست میتوانیم بگیم B x = که این هم تعداد میدهد مثل دنباله ولی اگر B نباشد undefined میدهد و غلط میشود واسه همین count رو تعریف کردیم که بگ میگیرد بعد یک فانکشن میدهد. # شبیه همان count هست که یک طرف بگ یک طرف x که بعد عدد طبیعی میدهد جا فانکشن.

فصل دهم:

برای یک تایپ اگر constructor بگذاریم با مجموعه مبدا یعنی اینکه میایم یک کپی از سورس را به عنوان فری تایپ تعریف میکنیم و اون مجموعه عناصر سورس به عنوان constant میشوند. این هم injective هست که سورس را مپ میکند به مقصد. برای ترتیب دهی اول میایم یک فری تایپ تعریف میکنیم که جای اون سازنده status هست که همین کار را میکند و degree به یک مجموعه دیگر ۰ تا ۳ مپ میشود انگار ۰ تا ۳ مپ میشود به مقادیر degree ها. بعد به عناصر degree نام درجات دانشگاهی را میدهیم.

| $ma = status\ 3$

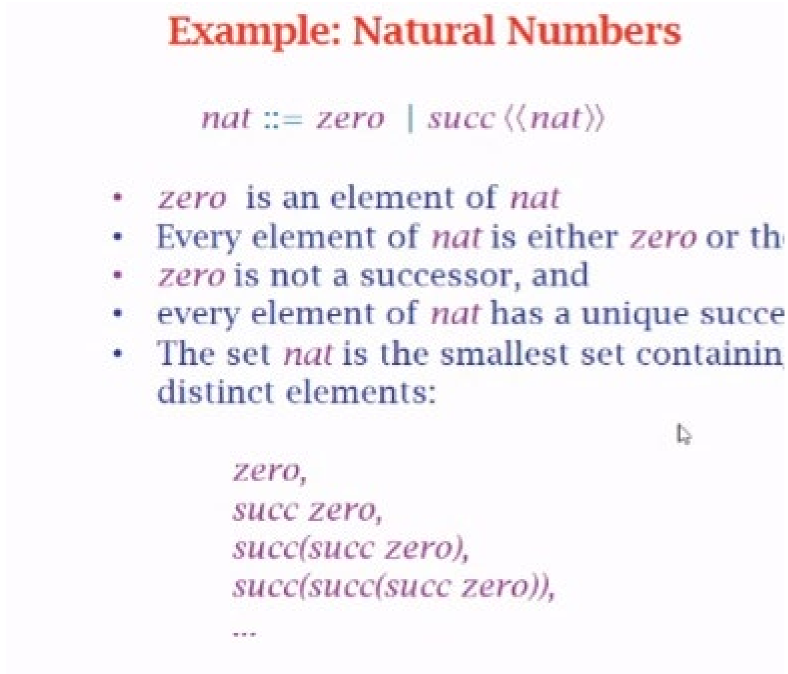
- We define the University's ordering of statuses of degrees $\leq_{status}$ in terms of the $\leq$ ordering on natural numbers:

$$\leq_{status} : Degree \longleftrightarrow Degree$$

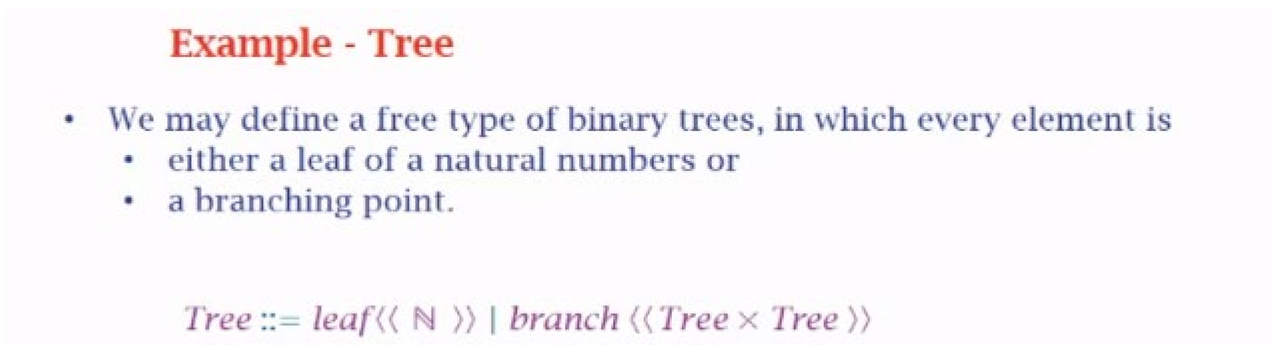
$$\forall\, d_1, d_2 : Degree \bullet$$
$$d_1 \leq_{status} d_2 \Leftrightarrow status^{\sim} d_1 \leq status^{\sim} d_2$$

بالا تعریف

رابطه پایین اجرا آن. فری تایپ میتواند هم constant باشد هم constructor داشته باشد.



هر اتم با n میشود یک لیست. Nil یک لیست هست atom 0 هم یک لیست هست cat atom 0 nill هم یک لیست میشود. atom 0 لزوما ترتیبی ندارد فقط یک لیست هست حالا هر ترتیبی. Nil یک constant هست.



دقت کن داخل <<>> هر چیزی نمیتوانی بگذاری باید مجموعه های محدود را بگذاری یا power خود مجموعه را بگذاری اینا باعث ناسازگاری میشوند. مشکل این هست که d یک فانکشن هست از power t به t و چون پاور از تی بزرگتر هست مپ یک به یک نمیتوانی داشته باشی.

اسلاید یازدهم:

Partial: یعنی صندلی سمت چپ بود به دو نفر نمیشود فروخت پس partial function هست.

بین decleration ها سمی کالون میگذاریم. اند هم باشد میتوانیم سطر به سطر بنویسیم.

Abstract data type یک مجموعه ای از متغیر با یک لیستی از عملیات هست مثل استک یا صف. متغیر ها را در یک شما بگذاریم میشود state سیستم ما و عملیات ها روی شما ها عمل میکنند و ما به حالت دیگری میرویم خود استیت ها قبل و بعد و عمل بین اینها را با شما تعریف میکنیم.

اسکیما هایی را میتوان مرج کرد که تعریف آنها یکسان باشد.

And میشود مرج declaration ها و and بین predicate ها. schema inclusion میشود همین and منتها اسم اسکیما را در محل declaration میگذاریم و predicate آن را در predicate اضافه میکنیم. Operation schema اومده state جفت ها رو برای حالت قبل و بعد آورده.

علامت سوال ورودی و علامت تعجب خروجی هست.

تتا همون characteristic binding هستش.