

Christine Rizkallah CSE, UNSW Term 3 2020

Composite Data Types

Most of the types we have seen so far are basic types, in the sense that they represent builten massing the sense that they represent builten massing that they represent the sense that the sense that they represent the sense that the sense that t

Real programming languages feature ways to *compose* types together to produce new types, such as:

https://powcoder.com

Add We Chat poweoder

Records

Combining values conjunctively

We want to store two things in one value.



Product Types

In Middle weighted the property of the complished have been property of the complished the complete type:

https://powwoder.com

We won't have type declarations, named fields or anything like that. More than two values can be combined by nexting trod lets for permissional vector:

 $\texttt{Int} \times (\texttt{Int} \times \texttt{Int})$

Constructors and Eliminators

we Assignment shirojectu Exam Help

https://poweoder.com

The only way to extract each component of the product is to use the fst and snd eliminators:

Add We Chat powcoder $\frac{\Gamma \vdash \text{fst } e : \tau_1}{\Gamma \vdash \text{snd } e : \tau_2}$

Examples

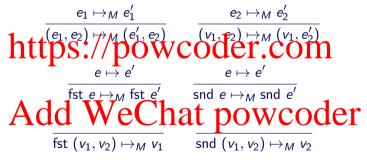
```
Example (Midpoint)

Assignment Project Exam Help
((\operatorname{Int} \times \operatorname{Int}) \to (\operatorname{Int} \times \operatorname{Int}) \to (\operatorname{Int} \times \operatorname{Int})) \ \rho_1 =
\operatorname{recfun\ midpoint'} ::
\operatorname{htt}((\operatorname{Int} \times /\operatorname{Int}) \to (\operatorname{Int} \times \operatorname{Int})) \ \rho_2 =
\operatorname{recfun\ midpoint'} ::
```

```
Example (Uncutried Division eChat powcoder recfun div :: ((Int \times Int)) \rightarrow Int) args = 
if (fst args < snd args)
then 0
else div (fst args - snd args, snd args)
```

Dynamic Semantics

Assignment Project Exam Help



Unit Types

Assignment Project Exam Help

Currently, we have no way to express a type with just one value. This may seem useless at first, but it becomes useful in combination with other types. We'll introduce at type Spronounce Wt, that as exact conclination, written ():

Add WeChat¹powcoder

Disjunctive Composition

We can't, with the types we have, express a type with exactly three values.

data Traffic Light = Red | Amber | Green | Exam Help

In general we want to express data that can be one of multiple alternatives, that contain different put to the contain different pu

```
Example (More elaborate alternatives)
```

This is awkward in many languages. In Java we'd have to use inheritance. In C we'd have to use unions.

Sum Types

We wall use sing types to express the possibility that data may be one of two forms.

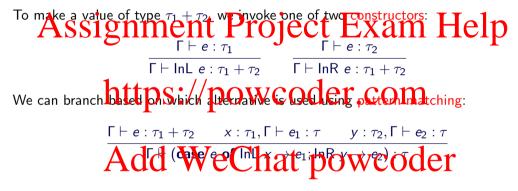
https://poweoder.com

This is similar to the Haskell Either type.

Our TrafficLight(1/16 called Extress Arbite (1/16 called Extress Arbi

 ${\tt TrafficLight} \simeq {\bf 1} + ({\bf 1} + {\bf 1})$

Constructors and Eliminators for Sums



(Using concrete syntax here, for readability.)
(Feel free to replace it with abstract syntax of your choosing.)

Examples

Assignment Project Exam Help

Our traffic light type has three values as required:

https://powcoder.com

Add we e Chat in Color Coder

Examples

We can convert most (non-recursive) Haskell types to equivalent MinHs types now.

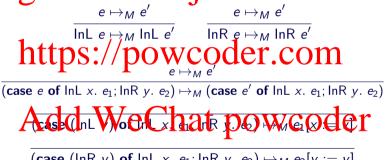
- Peplace ill constructors with Project Exam Help
- lacktriangledown Change the | character that separates constructors to a +.

tata Shape | Circle Length | Point | | Triangle Angle Length Length | | Add WeChat powcoder

```
egin{array}{lll} & 1 	imes (	ext{Int} 	imes 	ext{Int}) \ + & 1 	imes 	ext{Int} \ + & 1 	imes (	ext{Int} 	imes (	ext{Int} 	imes 	ext{Int})) \end{array}
```

Dynamic Semantics

Assignment Project Exam Help



(case (lnR v) of lnL x. e_1 ; lnR y. e_2) $\mapsto_M e_2[y:=v]$

The Empty Type

We add another than that has no invariants. Examines the light is no way to construct it.

We do have a way to eliminate it, however:

https://powcoder.com

 $\Gamma \vdash \text{absurd } e : \tau$

If I have a variable of the environment of the envi

Semiring Structure

These types we have defined form an algebraic structure called a *commutative*

SemirAge Signment Project Exam Help

- Associativity: $(\tau_1 + \tau_2) + \tau_3 \simeq \tau_1 + (\tau_2 + \tau_3)$
- Identity: $0+\tau \simeq \tau$ Commutate $0+\tau \simeq \tau$ Commutate $0+\tau \simeq \tau$

Laws for $(\tau, \times, \mathbf{1})$

- Associativity: $(\tau_1 \times \tau_2) \times \tau_3 \simeq \tau_1 \times (\tau_2 \times \tau_3)$
- Identity: 1A τ dd We Chat powcoder Commutativity: $\tau_1 \times \tau_2 = \tau_2 \times \tau_1$

Combining \times and +:

- Distributivity: $\tau_1 \times (\tau_2 + \tau_3) \simeq (\tau_1 \times \tau_2) + (\tau_1 \times \tau_3)$
- Absorption: $\mathbf{0} \times \tau \simeq \mathbf{0}$

What does \sim mean here?

Isomorphism

Two types τ_1 and τ_2 are *isomorphic*, written $\tau_1 \simeq \tau_2$, if there exists a *bijection* between them This means that for each value in τ_1 we can find a unique value in τ_2 and τ_3 we can use isomorphisms to simplify our Shape type:

isomorphisms to simplify our shape type.

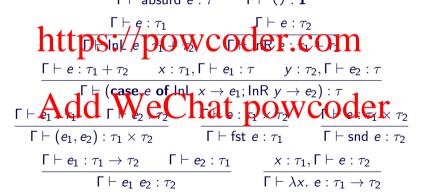
Add WeChat powcoder

$$\begin{array}{ll} & \texttt{Int} \times \texttt{Int} \\ + & \texttt{Int} + \mathbf{1} \\ + & \texttt{Int} \times (\texttt{Int} \times \texttt{Int}) \end{array}$$

Examining our Types

Lets look at the rules for typed lambda calculus extended with sums and products:

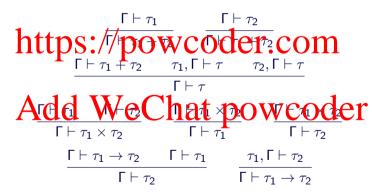
Assignment-Project Exam Help



Squinting a Little

Lets remove all the terms, leaving just the types and the contexts:

Assignment Project Exam Help

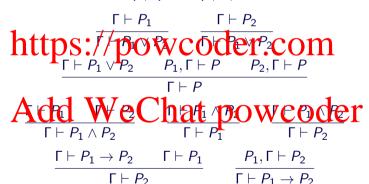


Does this resemble anything you've seen before?

A surprising coincidence!

Types are exactly the same structure as *constructive logic*:

Assignment Project Exam Help



This means, if we can construct a program of a certain type, we have also created a constructive proof of a certain proposition

The Curry-Howard Isomorphism

This correspondence goes by many names, but is usually attributed to Haskell Curry and William Howard ment Project Exam Help

	Programming	Logic	
1-44	Types	Propositions	
nup	S Prøgans V	COGG.CO	m
1	Evaluation	Propositions / COP of . CO Proof Simplification	

It turns out, no matter what logic you want to define, there is always a corresponding λ -calculus, and we were λ -calculus, and we were λ -calculus, and λ -calculus, an

Constructive Logic	Typed λ -Calculus
Classical Logic	Continuations
Modal Logic	Monads
Linear Logic	Linear Types, Session Types
Separation Logic	Region Types

Examples

Examples de la constant de la consta

and Comm :: $A \times B \rightarrow B \times A$ and Comm p = (snd p, fst p)This proves A **https:**//powcoder.com

Example (Transitivity of Implication)

AdditiWe Chat powcoder transitive f g x = g (f x)

Transitivity of implication is just function composition.

Caveats

All functions we define have to be total and terminating.

Otherwise we get an inconsistent logic that lets us prove false things: Help proof 1:: P = NP

 $proof_1 = proof_1$

https://powcoder.com

 $proof_2 :: P \neq NP$ $proof_2 = proof_2$

Most common calculi correspond to constructive logic, not classical ones, so principles like the law of excluded middle or double negation elimination do not hold:

 $\neg\neg P \rightarrow P$

Inductive Structures

What About types like lists? Project Exam Help

We can't express these in MinHS yet:

https://powcoder.com
Add WeChat powcoder

We need a way to do recursion!

Recursive Types

Assignment Project Exam Help

We introduce a new form of type, written $\mathbf{rec}\ t$. τ , that allows us to refer to the entire type:

Typing Rules

We desire general with Policy Goods to the xearm of the light unroll:

https://poweoder.com

Add WeChat, powcoder

 $\overline{\Gamma \vdash \mathsf{unroll}\ e : \tau[t := \mathsf{rec}\ t.\ \tau]}$

Example

Assignment Project Exam Help

```
Example
```

```
https://powdoder.com
```

```
 \begin{array}{ll} &=& \text{roll (lnL ())} \\ A_{t}d_{t}d_{t} &=& \text{Will Gallinfpower} \end{array}
```

Dynamic Semantics

Assignment Project Exam Help

$$https://powoder.com$$

Add WeChat powcoder