As for automata and Petri nets, we call a concurrent computation specification in combinatory logic a labeled arrow:

\[ u \xrightarrow{\alpha} v \]

That is, provable according to the inference system \( L(\text{CL}) \), i.e., we can build a proof tree for it as the root using the inference rules of \( L(\text{CL}) \). This gives as a labeled directed graph:

\[ \text{Spec} (\text{CL}) = (\text{CL}, \text{ProofTerms}_\text{CL}, \xrightarrow{\text{Spec} (\text{CL})}) \]

where \( \text{CL} \) are the CL expressions, \( \text{ProofTerms} \) are the \( \text{L} \) such that there are \( u, v \in \text{CL} \) such that \( u \xrightarrow{\alpha} v \) is a concurrent computation specification, and \( u \xrightarrow{\text{Spec} (\text{CL})} v \) iff \( u \xrightarrow{\alpha} v \) conc.comp.spec.

Note that arrows in this graph enjoy two operations:

1. **Application**: \( (u \xrightarrow{\alpha} u', v \xrightarrow{\beta} v') \xrightarrow{\text{Spec} (\text{CL})} u \circ v \xrightarrow{\alpha \beta} u' \circ v' \)

2. **Composition**: \( (u \xrightarrow{\alpha} u', v \xrightarrow{\beta} v) \xrightarrow{\text{Spec} (\text{CL})} u \circ v \xrightarrow{\alpha \beta} w \)
These two operations exist and are well defined precisely because of the Congruence and Transitivity inference rules. Note that \( CL \subseteq \text{PseudoTCL} \), and that \( -; - \) is defined on both, and both operations coincide on \( CL \) (\( CL \) is a subalgebra).

As before, the million-dollar question is: When are two \( CL \) computations specs the same?

or, what is equivalent:

What is a natural notion of \( CL \) computation?

By "engineering induction" based on our previous experience with automata and Petri nets, we can safely guess that:

1. **Associativity of \(-; -\):** Given \( U \xrightarrow{\alpha} V \xrightarrow{\beta} W \xrightarrow{\delta} q \)
   \[
   (\alpha; \beta)q \equiv U \xrightarrow{\alpha; (\beta; \delta)} q
   \]

2. **Identity:**
   \( U \xrightarrow{\alpha} V \equiv U \xrightarrow{\alpha} U \xrightarrow{\alpha} V \)

3. **Functionality:** Given also \( U' \xrightarrow{\alpha'} V' \xrightarrow{\beta'} W' \)
   \[
   (\alpha \alpha') \equiv U U' \xrightarrow{(\alpha; \beta) (\alpha'; \beta')} WW'
   \]
are all natural identities that hold between concurrent CL computations. What else? In CL we have encountered a new phenomenon that did not arise either for automata or for Petri nets, namely, the phenomenon of parametric transitions such as:

\[ k\text{-red}: (K x)y \rightarrow x \]

which are parametric on \( x, y \). That is, for any \( u, v \in \text{CL} \) we get a concrete basic transition

\[ k\text{-red}(u, v): (K u)v \rightarrow u \]

So, the natural question to ask is:

Are there natural identities between CL computations associated to the parametric transitions \( k\text{-red}, s\text{-red}, \text{I-red} \)?

Looking at a concrete example may give us some hints. Consider the computation specification:

\[ (((S ((K u)v))(I u))(Iv)) \rightarrow ((uv)(uv)) \]

which can be naturally decomposed as follows:

\[ \text{in page 5 of Lecture 7a} \]
\[
\begin{align*}
&((S((Ku)v))(Iu))(Iv) \\
&\quad \xrightarrow{S\text{-red}(Ku,v,Iu,Iv)} (((Ku)v)(Iu))(Iv) \\
&\quad \xrightarrow{S\text{-red}(u,v)} (u)(v) \\
&\quad \xrightarrow{(K\text{-red}(u,v)Ired(u)Ired(v))} (Ired(u))(Ired(v)) \\
&\quad \xrightarrow{Ired(v)} Ired(v) \\
&(S(K\text{-red}(u,v)Ired(u)Ired(v))(u) \\
&\quad \xrightarrow{(S\text{-red}(u,u,v))} (u)(v) \\
&\quad \xrightarrow{(S\text{-red}(u,v)} (u)(v) \\
&\quad \xrightarrow{(S\text{-red}(u,u,v))} (u)(v) \\
&\quad \xrightarrow{(S\text{-red}(u,v)} (u)(v)
\end{align*}
\]

That is, we can decompose the original computation by either:

**Upper Decomposition**

1. Applying \( S\text{-red} \) and then
2. Applying \( K\text{-red}, Ired(u) \) and \( Ired(v) \) in parallel (Congruence)

**Lower Decomposition**

1. Applying \( K\text{-red}, Ired(u), Ired(v) \) and then
2. Applying \( S\text{-red} \) in parallel (Congruence)

Of course, the way in which \( K\text{-red}, Ired(u) \) and \( Ired(v) \) are applied in parallel (Congruence) in the upper and lower cases is different, i.e., according to the terms \((Sx)y)z \) and \((xz)(yz)\) in

\[
S\text{-red}: ((Sx)y)z \rightarrow (xz)(yz)
\]
So, what does this suggest? The following:

For any \( u \xrightarrow{\alpha} u', \; v \xrightarrow{\beta} v', \; w \xrightarrow{\gamma} w' \) CL comp. specs

we have:

\[
\begin{align*}
((S \; u) \; v) \; w & \xrightarrow{S\text{-red}(u,v,w)} (u \; w) \; (v \; w) \\
(S \; (\alpha) \; (\beta)) \; (\gamma) & \equiv (\alpha \; (\beta)) \; (\gamma) \\
((S \; u') \; v') \; w' & \xrightarrow{S\text{-red}(u,v,w)} (u' \; w') \; (v' \; w')
\end{align*}
\]

or, more briefly:

4. \( ((S (\alpha) \; (\beta)) \; (\gamma)) ; \ S\text{-red}(u', v', w') \equiv S\text{-red}(\alpha, \beta, \gamma) \equiv S\text{-red}(u, v, w) ((\alpha \; (\beta)) (\gamma)) \)

For the exact same reason for \( K\text{-red} \) we get:

5. \( ((K \alpha) \; (\beta)) ; \ K\text{-red}(u', v') \equiv K\text{-red}(\alpha, \beta) \equiv K\text{-red}(u, v) ; \alpha \)

As for \( I\text{-red} \) we get:

6. \( (I \alpha) ; \ I\text{-red}(u') \equiv I\text{-red}(\alpha) \equiv I\text{-red}(u) ; \alpha \)

All the identifications (1)-(6) are completely natural and gives a natural notion of CL parallel computation:
Definition (Category $\mathcal{F}_{CL}$ of CL-computations).

The category $\mathcal{F}_{CL} = (CL, \text{PropTerm}_{CL}/\equiv, \rightarrow_{\mathcal{F}_{CL}})$ is defined by:

1. Identifying proof terms according to the smallest congruence generated by the equivalences (1)-(6) in the labeled graph $\text{Spec}(CL)$ with the two arrow operations of application and composition, is called the category of parallel CL computations.

That $\mathcal{F}_{CL}$ is a category and (ii) defining:

$$U \xrightarrow{[\alpha]_{\equiv}} V \iff U \xrightarrow{\alpha} V \text{ in spec. comp.}$$

is called the category of parallel CL computations.

That $\mathcal{F}_{CL}$ is a category follows immediately from the equivalences (1)-(2). But what other properties does $\mathcal{F}_{CL}$ have?

Since $\equiv$ denotes the congruence generated by (4)-(6), this means that it respects application and composition,
Since it respects composition we of course have that $\equiv_{TCL}$ is a category. But what does it mean that $\equiv_{TCL}$ respects application? It means that if

$$(u \xrightarrow{\alpha} u') \equiv (u \xrightarrow{\alpha'} u')$$

and

$$(v \xrightarrow{\beta} v') \equiv (v \xrightarrow{\beta'} v')$$

then we must have:

$$(uv \xrightarrow{\alpha \beta} uv') \equiv (uv \xrightarrow{\alpha' \beta'} uv').$$

But this exactly means that in $TCL$ we also have, as in $\text{Spec}(CL)$ an application operation on arrows:

$$(u \xrightarrow{[\alpha]} u', v \xrightarrow{[\beta]} v') \mapsto (uv \xrightarrow{[\alpha \beta]} uv')$$

What other properties does $TCL$ enjoy?

**Question:** To answer this property, we need to consider the concept of a natural transformation. What is that? To answer the second question we need to consider the concept of a functor. What is that? To answer this question we just need to first consider the hopefully familiar concept of (labeled) graph isomorphism.
Definition. Given labeled directed graphs $G = (N, L, \rightarrow)$ and $G' = (N', L', \rightarrow')$, a (labeled) graph-homomorphism $F: G \rightarrow G'$ is a function $F: N \rightarrow N'$ on nodes and $F: L \rightarrow L'$ on labels (we use the same notation for both for simplicity) such that:

\[
\text{If } n \xrightarrow{\ell} n' \text{ then } F(n) \xrightarrow{F(\ell)} F(n')
\]

Definition. Given categories $(A) = (O, A, \rightarrow)$ and $(A') = (O', A', \rightarrow')$ a functor $F: A \rightarrow A'$ is a graph-homomorphism $F: A \rightarrow A'$ such that:

1. Preserves composition, i.e., if $u \xrightarrow{\alpha} v \xrightarrow{\beta} w$, then $F(u) \xrightarrow{F(\alpha; \beta)} F(w) = (F(u) \xrightarrow{F(\beta)} F(v)) = F(u) \xrightarrow{F(\beta)} F(v) F(w)$

2. Preserves identities: $(F(u) \xrightarrow{F(id_u)} F(u)) = (F(u) \xrightarrow{id'_{F(u)}} F(u))$

were $-$ and $id'$ denote compose and identity in $A'$.
Let us some examples of functors:

1. $U : \text{CommMon} \rightarrow \text{Set}$ forgets about the monoid structure, i.e.:

$$U((M, +, o) \xrightarrow{f} ((M', +', o')) = M \xrightarrow{f} M$$

2. The **categorical product of sets** is also a functor:

$$- \times - : \text{Set} \times \text{Set} \rightarrow \text{Set}$$

$$(A \xrightarrow{f} A') \times (B \xrightarrow{g} B') = A \times B \xrightarrow{f \times g} A' \times B'$$

where we define:

$$f \times g = \text{def} \lambda (a, b) \in A \times B. (f(a), g(b)) \in A' \times B'$$

But what is the category $\text{Set} \times \text{Set}$?

Definition. Given categories $A = (O, A, \rightarrow_A)$ and $A' = (O', A', \rightarrow_{A'})$, $A \times A'$ denote the category: $(A \times A') = (O \times O', A \times A', \rightarrow_{A \times A'})$

where:

$$(m, n') \xrightarrow{(f, g')}{A \times A'} (m', n') \text{ iff } n \xrightarrow{f} m \text{ and } n' \xrightarrow{g'} m'$$

- $\text{id}_{(m, n')} = (\text{id}_m, \text{id}_{n'}) \text{ for any } (m, n') \in O \times O'$

- If $(m, n') \xrightarrow{(f, g')} (m, n')' \xrightarrow{(k, k')}{A \times A'} (k', k'')$, then
\[
\begin{align*}
(n, m') & \quad \xrightarrow{(f, f')} (g, g') \\
A \times A
\end{align*}
\]

3. Therefore, for any Petri net the fact that the union operation acts on the arrows of \( \mathcal{N} \) is just the fact that \( \_ \circ_\_ \) is a functor.

\[
\begin{align*}
\circ \quad : \quad \mathcal{N} \times \mathcal{N} & \quad \rightarrow \quad \mathcal{N} \\
((u, v) \xrightarrow{\alpha, \beta} (u', v')) & \quad \mapsto \quad (u u' \xrightarrow{\alpha \beta} v v')
\end{align*}
\]

4. Likewise for CL logic application is a functor.

\[
\begin{align*}
\circ \quad : \quad \mathcal{N}_{cl} \times \mathcal{N}_{cl} & \quad \rightarrow \quad \mathcal{N}_{cl} \\
((u, v) \xrightarrow{\alpha, \beta} (u', v')) & \quad \mapsto \quad (u u' \xrightarrow{\alpha \beta} v v')
\end{align*}
\]

**Notation.** For a set, a category, \( A^n = \text{A}_\times \times \text{A} \), and \( A^n = \text{A}_\times \times \times \text{A} \).

5. For any \( n \) (assuming \( A^n = \{1\} \) (one-point set), and \( A^1 = \text{A} \)) we also have a functor.

\[
(\_ \circ_\_)^n : \quad \text{Set} \quad \rightarrow \quad \text{Set}
\]

\[
(\text{A} \xrightarrow{f} \text{B}) \quad \mapsto \quad (\text{A}^n \xrightarrow{f^n} \text{B}^n)
\]