15-819K: Logic Programming

Lecture 12

Linear Logic

Frank Pfenning

October 5, 2006

In this lecture we will rewrite the program for peg solitaire in a way that treats state logically, rather than as an explicit data structure. In order to allow this we need to generalize the logic to handle state intrinsically, something provided by *linear logic*. We provide an introduction to linear logic as a *sequent calculus*, which generalizes our previous way of specifying truth. The sequent calculus is a bit too general to allow an immediate operational interpretation to obtain a logic programming language, so we postpone this step to the next lecture.

12.1 State-Passing Style

Let us reexamine the central part of the program for peg solitaire from the last lecture. The first predicate `moves` passes state downward in the program.

```
moves(0, _).
moves(N, State1) :-
        N > 0,
        move1(State1, State2),
        N1 is N-1,
        moves(N1, State2).
```

The second and third predicates, `move1` and `swap` pass state from an input argument (the first) to an output argument (the last).

```
move1(State1, State4) :-
        ( between(A,B,C) ; between(C,B,A) ),
        swap(State1, peg(A), State2),
        swap(State2, peg(B), State3),
        swap(State3, hole(C), State4).

swap([peg(A)|State], peg(A), [hole(A)|State]).
swap([hole(A)|State], hole(A), [peg(A)|State]).
swap([Place|State1], Place0, [Place|State2]) :-
        swap(State1, Place0, State2).
```

This pattern of code is called *state-passing* or *store-passing*. State-passing style is a common pattern in logic programs of a certain kind, specifically state exploration as in puzzles and games or modeling concurrent or distributed systems.

We investigate in this and the next lecture how to eliminate explicit state passing from such programs in favor of logical primitives.

In functional programming, the related *store-passing* style usually arises in the opposite way: if we want to turn a functional program that uses mutable storage into a pure functional program we can pass the store around as an explicit argument.

12.2 State-Dependent Truth

In our program, state is represented as a list of items `peg(`ij`)` and `hole(`ij`)` for locations ij. Stepping back from this particular representation, it is easy to interpret `peg` and `hole` as *predicates*, and `peg(`ij`)` and `hole(`ij`)` as *propositions*. For example, we say the proposition `peg(`ij`)` is true if there is a peg in location ij on the board.

What makes this somewhat unusual, from the perspective of the logic we have considered so far, is that the notion of truth depends on the state. In some states, `peg(`ij`)` is true, in some it is false. In fact, the state of the board is completely characterized by the `peg` and `hole` propositions.

In mathematical logic, truth is normally invariant and does depend on state. This is because the mathematical objects we deal with, such as natural numbers, are themselved invariant and considered universal. In philosophical logic, however, the concept of truth depending on the state of the world is central and has been investigated under the name *modal logic*, of which *temporal logic* is a particular branch. In these logics truth explicitly depends on the state of the world, and the separate concept of necessary

truth captures those properties that are state invariant. However, neither modal nor temporal logic is particularly appropriate for our problem domain. As an example, let us consider how we would specify a legal state transition, using the next-time operator $\bigcirc$. We might write

$$\begin{array}{c}\mathsf{between}(A,B,C) \wedge \mathsf{peg}(A) \wedge \mathsf{peg}(B) \wedge \mathsf{hole}(C) \\ \supset \bigcirc(\mathsf{hole}(A) \wedge \mathsf{hole}(B) \wedge \mathsf{hole}(C)).\end{array}$$

Unfortunately, unless we specify something else, the *only* thing we know about the next state is $\mathsf{hole}(A) \wedge \mathsf{hole}(B) \wedge \mathsf{hole}(C)$. What we would really like to say is that all other propositions regarding locations besides A, B, and C remain unchanged.

This kind of circumscription is awkward, defeating the purpose of obtaining a higher-level and more elegant formulation of our example and similar state-passing code. Moreoever, when we add more predicates then the move specification must also change to carry these over unchanged. In artificial intelligence this is called the *frame problem*. In the next section we show an elegant and completely logical solution to this problem.

12.3 Linear Logic

Linear logic has been described as a logic of state or a resource-aware logic.[1] Formally, it arises from complementing the usual notion of logical assumption with so-called *linear assumptions* or *linear hypotheses*. Unlike traditional assumptions which may be used many times in a proof, linear assumptions must be used *exactly once* during a proof. Linear assumptions then become (consumable) *resources* in the course of a proof.

This generalization of the usual mathematical standpoint may seem slight, but as we will see it is quite expressive. We write

$$A_1\ \mathit{res}, \ldots, A_n\ \mathit{res} \Vdash C\ \mathit{true}$$

for a linear hypothetical judgment with resources $A_1, \ldots, A_n$ and goal C. If we can prove this, it means that we can achieve that C is true, given resources A_1 through A_n. Here, all A_i and C are propositions.[2] The version of linear logic defined by this judgment is called *intuitionistic linear logic*,

[1] The term *linear* is connected to its use in algebra, but the connection is not easy to explain. For this lecture just think of "*linear*" as denoting "*must be used exactly once*".

[2] In the end it will turn out that $A\ \mathit{res}$ and $A\ \mathit{true}$ are interchangeable in that we can go from each one to the other. At this point, however, we do not know this yet, so the judgment we make about our resources is not that they are true, but that they are given resources.

sometimes contrasted with *classical linear logic* in which the sequent calculus has multiple conclusions. While it is possible to develop classical linear logic programming it is more difficult to understand and use.

Hidden in the judgment are other assumptions, usually abbreviated as Γ, which can be used arbitrarily often (including not at all), and are therefore called the *unrestricted assumptions*. If we need to make them explicit in a rule we will write

$$\Gamma; \Delta \Vdash C \ true$$

where Δ abbreviates the resources. As in our development so far, unrestricted assumption are fixed and are carried through from every conclusion to all premisses. Eventually, we will want to generalize this, but not quite yet.

The first rule of linear logic is that if we have a resource P we can achieve goal P, where P is an atomic proposition. It will be a consequence of our definitions that this will be true for arbitrary propositions A, but we need it as a rule only for the atomic case, where the structure of the propositions can not be broken down further.

$$\frac{}{P \ res \Vdash P \ true} \ \mathrm{id}$$

We call this the *identity rule*, it is also sometimes called the *init* rule, and the sequent $P \Vdash P$ is called an *initial sequent*.

12.4 Connectives of Linear Logic

One of the curious phenomena of linear logic is that the ordinary connectives multiply. This is because the presence of linear assumptions allows us to make distinctions we ordinarily could not. The first example of this kind is conjunction. It turns out that linear logic possesses two forms of conjunction.

Simultaneous Conjunction ($A \otimes B$). A simultaneous conjunction $A \otimes B$ is true if we can achieve both A and B in the same state. This means we have to subdivide our resources, devoting some of them to achieve A and the others to achieve B.

$$\frac{\Delta = (\Delta_A, \Delta_B) \quad \Delta_A \Vdash A \quad \Delta_B \Vdash B}{\Delta \Vdash A \otimes B} \otimes R$$

The order of linear assumptions is irrelevant, so in $\Delta = (\Delta_A, \Delta_B)$ the comma denotes the multi-set union. In other words, every occurrence of a proposition in Δ will end up in exactly one of Δ_A and Δ_B.

If we name the initial state of peg solitaire Δ_0, then we have $\Delta_0 \Vdash \mathsf{peg}(33) \otimes \mathsf{hole}(03) \otimes \ldots$ for some "..." because we can achieve a state with a peg at location 33 and hole at location 03. On the other hand, we cannot prove $\Delta_0 \Vdash \mathsf{peg}(33) \otimes \mathsf{hole}(33) \otimes \ldots$ because we cannot have a peg and an empty hole at location 33 in the same state. We will make the ellipsis "..." precise below as consumptive truth $\top$.

In a linear sequent calculus, the right rules shows when we can conclude a proposition. The left rule shows how we can use a resource. In this case, the resource $A \otimes B$ means that we have A and B simultaneously, so the left rule reads

$$\frac{\Delta, A\ res, B\ res \Vdash C\ true}{\Delta, A \otimes B\ res \Vdash C\ true}\ \otimes L.$$

In comparison to the focusing judgment we used to explain the logical semantics of pure Prolog programs, the left rules are not restricted to continuously decompose a single proposition until an atomic form is reached. Instead, various applicable left rules that operate on different assumptions can be freely interleaved. We consider the restriction to focusing in the next lecture.

Alternative Conjunction ($A \mathbin{\&} B$). An alternative conjunction is true if we can achieve both conjuncts, separately, with the current resources. This means if we have a linear assumption $A \mathbin{\&} B$ we have to make a choice: either we use A or we use B, but we cannot use them both.

$$\frac{\Delta \Vdash A\ true \quad \Delta \Vdash B\ true}{\Delta \Vdash A \mathbin{\&} B\ true}\ \&R$$

$$\frac{\Delta, A\ res \Vdash C\ true}{\Delta, A \mathbin{\&} B\ res \Vdash C\ true}\ \&L_1 \qquad \frac{\Delta, B\ res \Vdash C\ true}{\Delta, A \mathbin{\&} B\ res \Vdash C\ true}\ \&L_2$$

It looks like the right rule duplicates the assumptions, but this does not violate linearity because in a use of the assumption $A \mathbin{\&} B\ res$ we have to commit to one or the other.

Returning to the solitaire example, we have $\Delta_0 \Vdash \mathsf{peg}(33) \otimes \mathsf{hole}(03) \otimes \ldots$ and we also have $\Delta_0 \Vdash \mathsf{hole}(33) \otimes \mathsf{hole}(03) \otimes \ldots$ because we can certainly reach states with these properties. However, we cannot reach a single state with both of these, because the two properties of location 33 clash. If we

want to express that both are reachable, we can form their alternative conjunction

$$\Delta_0 \Vdash (\mathsf{peg}(33) \otimes \mathsf{hole}(03) \otimes \ldots) \,\&\, (\mathsf{hole}(33) \otimes \mathsf{hole}(03) \ldots).$$

Consumptive Truth ($\top$). We have seen two forms of conjunction, which are distinguished because of their resource behavior. There are also two truth constants, which correspond to zero-ary conjunctions. The first is *consumptive truth* $\top$. A proof of it consumes all current resources. As such we can extract no information from its presence as an assumption.

$$\frac{}{\Delta \Vdash \top\ \mathit{true}}\ \top R \qquad\qquad \begin{array}{c}\text{no } \top L \text{ rule} \\ \Delta, \top\ \mathit{res} \Vdash C\ \mathit{true}\end{array}$$

Consumptive truth is important in applications where there is an aspect of the state we do not care about, because of the stipulation of linear logic that every linear assumption must be used *exactly once*. In the examples above so far we cared about only two locations, 33 and 03. The state will have a linear assumption for every location, which means we can *not* prove, for example, $\Delta_0 \Vdash \mathsf{peg}(33) \otimes \mathsf{hole}(03)$. However, we *can* prove $\Delta_0 \Vdash \mathsf{peg}(33) \otimes \mathsf{hole}(03) \otimes \top$, because the consumptive truth matches the remaining state.

Consumptive truth is the unit of alternative conjunction in that $A \,\&\, \top$ is equivalent to A.

Empty Truth (1). The other form of truth holds only if there are no resources. If we have this as a linear hypothesis we can transform it into the empty set of resources.

$$\frac{\Delta = (\cdot)}{\Delta \Vdash \mathbf{1}\ \mathit{true}}\ \mathbf{1}R \qquad\qquad \frac{\Delta \Vdash C\ \mathit{true}}{\Delta, \mathbf{1}\ \mathit{res} \Vdash C\ \mathit{true}}\ \mathbf{1}L$$

Empty truth can be useful to dispose explicitly of specific resources.

Linear Implication ($A \multimap B$). A linear implication $A \multimap B$ is true if we can achieve B given resource A.

$$\frac{\Delta, A\ \mathit{res} \Vdash B\ \mathit{true}}{\Delta \Vdash A \multimap B\ \mathit{true}}\ \multimap R$$

Conversely, if we have $A \multimap B$ as a resource, it means that we could transform the resource A into the resource B. We capture this in the following left rule:

$$\frac{\Delta = (\Delta_A, \Delta_B) \quad \Delta_A \Vdash A\ true \quad \Delta_B, B\ res \Vdash C\ true}{\Delta, A \multimap B\ res \Vdash C\ true} \multimap L.$$

An assumption $A \multimap B$ therefore represents a means to transition from a state with A to a state with B.

Unrestricted Assumptions Γ. The left rule for linear implication points at a problem: the linear implication is itself linear and therefore consumed in the application of that rule. If we want to specify via a linear logic program how state may change, we will need to reuse the clauses over and over again. This can be accomplished by a *copy* rule which takes an unrestricted assumption and makes a linear copy of it. It is actually very much like the focusing rule in an earlier system.

$$\frac{A\ ures \in \Gamma \quad \Gamma; \Delta, A\ res \Vdash C\ true}{\Gamma; \Delta \Vdash C\ true}\ \text{copy}$$

We label the unrestricted assumptions as unrestricted resources, $A\ ures$. In the logic programming interpretation, the whole program will end up in Γ as unrestricted assumptions, since the program clauses can be used arbitrarily often during a computation.

Resource Independence ($!A$). The proposition $!A$ is true if we can prove A without using any resources. This means we can produce as many copies of A as we need (since it costs nothing) and a linear resource $!A$ licenses us to make the unrestricted assumption A.

$$\frac{\Gamma; \cdot \Vdash A\ true}{\Gamma; \cdot \Vdash !A\ true}\ !R \qquad \frac{(\Gamma, A\ ures); \Delta \Vdash C\ true}{\Gamma; \Delta, !A\ res \Vdash C\ true}\ !L$$

Disjunction ($A \oplus B$). The familiar conjunction from logic was split into two connectives in linear logic: the simultaneous and the alternative conjunction. Disjunction does not split the same way unless we introduce an explicit judgment for falsehood (which we will not pursue). The goal $A \oplus B$ can be achieved if we can achieve either A or B.

$$\frac{\Delta \Vdash A\ true}{\Delta \Vdash A \oplus B\ true} \oplus R_1 \qquad \frac{\Delta \Vdash B\ true}{\Delta \Vdash A \oplus B\ true} \oplus R_2$$

Conversely, if we are given $A \oplus B$ as a resource, we do not know which of the two is true, so we have to account for both eventualities. Our proof splits into cases, and we have to show that we can achieve our goal in either case.

$$\frac{\Delta, A\ res \Vdash C\ true \quad \Delta, B\ res \Vdash C\ true}{\Delta, A \oplus B\ res \Vdash C\ true}\ \oplus L$$

Again, it might appear as if linearity is violated due to the duplication of Δ and even C. However, only one of A or B will be true, so only one part of the plan represented by the two premisses really applies, preserving linearity.

Falsehood (0). There is no way to prove falsehood $\mathbf{0}$, so there is no right rule for it. On the other hand, if we have $\mathbf{0}$ as an assumption we know we are really in an impossible state so we are permitted to succeed.

$$\begin{array}{l}\text{no } \mathbf{0}R \text{ rule} \\ \Delta \Vdash \mathbf{0}\ true\end{array} \qquad \frac{}{\Delta, \mathbf{0}\ res \Vdash C\ true}\ \mathbf{0}L$$

We can also formally think of falsehood as a disjunction between zero alternatives and arrive at the same rule.

12.5 Resource Management

The connectives of linear logic are generally classified into *multiplicative*, *additive*, and *exponential*.[3]

The multiplicative connectives, when their rules are read from conclusion to the premisses, split their resources between the premisses. The connectives $\otimes$, $\mathbf{1}$, and $\multimap$ have this flavor.

The additive connectives, when their rules are read from conclusion to premisses, propagate their resources to all premisses. The connectives $\&$, $\top$, $\oplus$, and $\mathbf{0}$ have this flavor.

The exponential connectives mediate the boundary between linear and non-linear reasoning. The connective ! has this flavor.

During proof search (and therefore in the logic programming setting), a significant question is how to handle the resources. It is clearly impractical, for example, in the rule

$$\frac{\Delta = (\Delta_A, \Delta_B) \quad \Delta_A \Vdash A\ true \quad \Delta_B \Vdash B\ true}{\Delta \Vdash A \otimes B\ true}\ \otimes R$$

[3] Again, we will not try to explain the mathematical origins of this terminology.

to simply enumerate all possibilities and try to prove A and B in each combination until one is found that works for both.

Instead, we pass in all resources Δ into the first subgoal A and keep track which resources are consumed. We then pass the remaining ones to the proof of B. Of course, if B fails we may have to find another proof of A which consumes a different set of resources and then retry B, and so on. In the logic programming setting this is certainly an issue the programmer has to be aware of, just as the programmer has to know which subgoal is solved first, or which clause is tried first.

We will return to this question in the next lecture where we will make resource-passing explicit in the operational semantics.

12.6 Peg Solitaire, Linearly

We now return to the peg solitaire example. We coule like to rewrite the `moves` predicate from a state-passing `moves(`n`,`s`)` to just `moves(`n`)`, where the state s is actually encoded in the linear context Δ. That is, we consider the sitation

$$\Delta \Vdash \mathsf{moves}(n)$$

where Δ contains a linear assumption $\mathsf{peg}(ij)$ when there is a peg in location ij and $\mathsf{hole}(ij)$ if there is an empty hole in location ij.

The first clause,

```
moves(0, _).
```

is translated to

```
moves(0) ⟜ ⊤.
```

where $\multimap$ is reverse linear implication. The $\top$ here is necessary to consume the state (which, in this case, we don't care about).

The second clause for `moves`

```
moves(N, State1) :-
        N > 0,
        move1(State1, State2),
        N1 is N-1,
        moves(N1, State2).
```

as well as the auxiliary predicates `move1` and `swap` are replaced by just one clause in the definition of `moves`.

```
moves(N) ⟜
        N > 0 ⊗ N1 is N-1 ⊗
        ( between(A,B,C) ⊕ between(C,B,A) ) ⊗
        peg(A) ⊗ peg(B) ⊗ hole(C) ⊗
        (hole(A) ⊗ hole(B) ⊗ peg(C) ⊸ moves(N1)).
```

Operationally, we first compute $n-1$ and then find a triple A, B, C such that B is between A and C. These operations are state independent, although the clause does not indicate that.

At this point we determine if there are pegs at A and B and a hole at C. If this is not the case, we fail and backtrack; if it is we remove these three assumptions from the linear context (they are consumed!) and assume instead $\mathsf{hole}(A)$, $\mathsf{hole}(B)$, and $\mathsf{peg}(C)$ before calling moves recursively with $n - 1$. At this point this is the only outstanding subgoal, and the state has changed by jumping A over B into C, as specified.

Observe how linearity and the intrinsic handling of state let's us replace a lot of code for state management with one short clause.

12.7 Historical Notes

Linear logic in a slightly different form than we present here is due to Girard [2]. He insisted on a classical negation in his formulation, which can get in the way of an elegant logic programming formulation. The judgmental presentation we use here was developed for several courses on *Linear Logic* [3] at CMU. Some additional connectives, and some interesting connections between the two formulations in linear logic are developed by Chang, Chaudhuri and Pfenning [1]. We'll provide some references on linear logic programming in the next lecture.

12.8 Exercises

Exercise 12.1 *Prove that $A\ res \Vdash A\ true$ for any proposition A.*

Exercise 12.2 *For each of the following purely linear entailments, give a proof that they hold or demonstrate that they do not hold because there is no deduction in our system. You do not need to prove formally that no deduction exists.*

i. $A \,\&\, (B \oplus C) \Vdash (A \,\&\, B) \oplus (A \,\&\, C)$

ii. $A \otimes (B \oplus C) \Vdash (A \otimes B) \oplus (A \otimes C)$

iii. $A \oplus (B \,\&\, C) \Vdash (A \oplus B) \,\&\, (A \oplus C)$

iv. $A \oplus (B \otimes C) \Vdash (A \oplus B) \otimes (A \oplus C)$

Exercise 12.3 *Repeat Exercise 12.2 by checking the reverse linear entailments.*

Exercise 12.4 *For each of the following purely linear entailments, give a proof that they hold or demonstrate that they do not hold because there is no deduction in our system. You do not need to prove formally that no deduction exists.*

i. $A \multimap (B \multimap C) \Vdash (A \otimes B) \multimap C$

ii. $(A \otimes B) \multimap C \Vdash A \multimap (B \multimap C)$

iii. $A \multimap (B \mathbin{\&} C) \Vdash (A \multimap B) \mathbin{\&} (A \multimap C)$

iv. $(A \multimap B) \mathbin{\&} (A \multimap C) \Vdash A \multimap (B \mathbin{\&} C)$

v. $(A \oplus B) \multimap C \Vdash (A \multimap C) \mathbin{\&} (A \multimap C)$

vi. $(A \multimap C) \mathbin{\&} (A \multimap C) \Vdash (A \oplus B) \multimap C$

Exercise 12.5 *For each of the following purely linear entailments, give a proof that they hold or demonstrate that they do not hold because there is no deduction in our system. You do not need to prove formally that no deduction exists.*

i. $C \Vdash \mathbf{1} \multimap C$

ii. $\mathbf{1} \multimap C \Vdash C$

iii. $A \multimap \top \Vdash \top$

iv. $\top \Vdash A \multimap \top$

v. $\mathbf{0} \multimap C \Vdash \top$

vi. $\top \Vdash \mathbf{0} \multimap C$

Exercise 12.6 *For each of the following purely linear entailments, give a proof that they hold or demonstrate that they do not hold because there is no deduction in our system. You do not need to prove formally that no deduction exists.*

i. $!(A \otimes B) \Vdash !A \otimes !B$

ii. $!A \otimes !B \Vdash !(A \otimes B)$

iii. $!(A \mathbin{\&} B) \Vdash !A \otimes !B$

iv. $!A \otimes !B \Vdash !(A \,\&\, B)$

v. $!\top \Vdash \mathbf{1}$

vi. $\mathbf{1} \Vdash !\top$

vii. $!\mathbf{1} \Vdash \top$

viii. $\top \Vdash !\mathbf{1}$

ix. $!!A \Vdash !A$

x. $!A \Vdash !!A$

12.9 References

[1] Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon University, December 2003.

[2] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[3] Frank Pfenning. Linear logic. Lecture Notes for a course at Carnegie Mellon University, 1995. Revised 1998, 2001.

12.10 Appendix: Summary of Intuitionistic Linear Logic

In the rules below, we show the unrestricted assumptions Γ only where affected by the rule. In all other rules it is propagated unchanged from the conclusion to all the premisses. Also recall that the order of hypotheses is irrelevant, and Δ_A, Δ_B stands for the multiset union of two collections of linear assumptions.

Judgmental Rules

$$\frac{}{P\ res \Vdash P\ true}\ \text{id} \qquad \frac{A\ ures \in \Gamma \quad \Gamma; \Delta, A\ res \Vdash C\ true}{\Gamma; \Delta \Vdash C\ true}\ \text{copy}$$

Multiplicative Connectives

$$\frac{\Delta_A \Vdash A \quad \Delta_B \Vdash B}{\Delta_A, \Delta_B \Vdash A \otimes B}\ \otimes R \qquad \frac{\Delta, A\ res, B\ res \Vdash C\ true}{\Delta, A \otimes B\ res \Vdash C\ true}\ \otimes L$$

$$\frac{}{\cdot \Vdash \mathbf{1}\ true}\ \mathbf{1}R \qquad \frac{\Delta \Vdash C\ true}{\Delta, \mathbf{1}\ res \Vdash C\ true}\ \mathbf{1}L$$

$$\frac{\Delta, A\ res \Vdash B\ true}{\Delta \Vdash A \multimap B\ true}\ \multimap R \qquad \frac{\Delta_A \Vdash A\ true \quad \Delta_B, B\ res \Vdash C\ true}{\Delta_A, \Delta_B, A \multimap B\ res \Vdash C\ true}\ \multimap L$$

Additive Connectives

$$\frac{\Delta \Vdash A\ true \quad \Delta \Vdash B\ true}{\Delta \Vdash A \,\&\, B\ true}\ \&R \qquad \frac{\Delta, A\ res \Vdash C\ true}{\Delta, A \,\&\, B\ res \Vdash C\ true}\ \&L_1 \qquad \frac{\Delta, B\ res \Vdash C\ true}{\Delta, A \,\&\, B\ res \Vdash C\ true}\ \&L_2$$

$$\frac{}{\Delta \Vdash \top\ true}\ \top R \qquad \text{no } \top L \text{ rule}$$

$$\frac{\Delta \Vdash A\ true}{\Delta \Vdash A \oplus B\ true}\ \oplus R_1 \qquad \frac{\Delta, A\ res \Vdash C\ true \quad \Delta, B\ res \Vdash C\ true}{\Delta, A \oplus B\ res \Vdash C\ true}\ \oplus L \qquad \frac{\Delta \Vdash B\ true}{\Delta \Vdash A \oplus B\ true}\ \oplus R_2$$

$$\text{no } \mathbf{0}R \text{ rule} \qquad \frac{}{\Delta, \mathbf{0}\ res \Vdash C\ true}\ \mathbf{0}L$$

Exponential Connective

$$\frac{\Gamma; \cdot \Vdash A\ true}{\Gamma; \cdot \Vdash\ !A\ true}\ !R \qquad \frac{(\Gamma, A\ ures); \Delta \Vdash C\ true}{\Gamma; \Delta, !A\ res \Vdash C\ true}\ !L$$

Figure 1: Intuitionistic Linear Logic