Background

The Ptolemy II Type System is similar to HM(X): Hindley-Milner over a constraint system.

Characteristics
Static typing, dynamic checking, type inference, subtyping, automatic type conversion, polymorphism, structured types

(Forward) Type Inference
- Actors have typed ports
- Types are inferred for ports that are left typed unknown
- Type inference is driven by type constraints that are imposed to guarantee that tokens are (forward-)compatible with i.e., lossless convertible to the types of their respective downstream destinations
- Roughly, this means that between actors $T_{\text{input}} \leq T_{\text{output}}$ and within actors $T_{\text{input}} \leq T_{\text{output}}$ where all types are ordered in a lattice.
- Prior to execution, all type constraints are harvested from the model, and a linear time (Rehof and Mogensen 1999) constraint solving algorithm is run to find a least fixed point that satisfies all constraints
- The solution is accepted if all constraints are satisfied and no types are left unknown

Problem

We want to build Ptolemy models that use online data, but such data is typically unreliable, subject to change, and most importantly, untyped.

```
“What’s the type?”
```

```
Figure 3: A Ptolemy model requires each port of every actor to be typed prior to execution, or otherwise, a type error is thrown. When an actor parses untyped data, the type of its output cannot be inferred.
```

Solution

Enable backward type inference.

The types for otherwise under-determined outputs are backward inferred based on type constraints imposed by downstream actors.

```
Figure 4: Dialog after right-clicking on the background of the composite actor in Vergil.
```

```
Example 2: The SequencePlotter accepts anything less than or equal to double, which through backward type constraints determines the output type of JSONToToken.
```

Goal

Maximally Permissive Composition

- Infer types that are specific enough not to limit composability yet general enough not to impose unnecessary constraints.

Robustness
- Let type safe operation of actors be guaranteed by the run-time type checker.
- Do error handling at the sending side to allow fall-back modes to kick in before the model comes to a grinding halt.

Convenience
- Automatically infer types to unburden the user. / programmer

Future

Ptolemy models that use web resources are still rather brittle: absence of an appropriate response to a request for remote data will raise a run-time type error, which brings execution to a halt.

By defining alternative error handling strategies, we can make models more robust, possibilities are:

- Retry
- Try another resource
- Resend a previous value
- Output a nil token
- Output no token at all

```
Figure 5: An error transition.
```

Modifications

We leverage type inference to statically type dynamic data and leverage dynamic type checking to invoke error handling strategies that enhance robustness.

```
Figure 2: we find a solution in area S, slightly higher than the original least fixed point.
```

```
Additional type constraints
- Backward constraint between actors: $\text{GLB}(T_{\text{output},\text{left}}) \leq T_{\text{output}}$
- Backward constraint within actors: $\text{GLB}(T_{\text{output,\text{left}}}) \leq T_{\text{output}}$ (simplified)
- Sinks actors: Input as general as possible

Implementation
- Toggle backward type inference per composite actor
- No impact on run-time of type resolution
- Type errors trapped at the source
- Suitable as activation mechanism for custom error handling

Example 1: The JSONToToken actor, by nature of what it does, cannot provide any specific information about its output.