Extensions to the Ptolemy II Type System

Yuhong Xiong  
Edward A. Lee

Type Lattice

- Specifies the lossless type conversion or subtype relations
Type Constraints

\[ \alpha \leq \gamma \]
\[ \beta \leq \gamma \]
\[ \gamma \leq \text{Complex} \]

Extensions

- Structured types
- Using monotonic functions in type constraints
Structured Types

Goals and Questions

- Arbitrary element types. E.g. (int)array, ((int)array)array, array of records, records containing arrays, ...
- Type constraints between element types and the types of other objects in system
- Order relation among structured types?
- Structured types admitted by the inequality solving algorithm?
- Convergence on infinite lattice?
Order Relation

- Array: order by element type:
  - (Int)array ≤ (Double)array

- Record: subtype
  
  \{item: String, val: Double\}

  \{item: String, val: Int\}    \{item: String, val: Double, id: Int\}

Inequality Solving Algorithm

- Admits (Rehof and Mogensen ’96):

  \[
  \text{const } \alpha \quad f(\alpha) \quad \leq \quad \text{const } \alpha
  \]

- Extension:

  (Int)array ≤ (\alpha)array

  know the “structure” of right-hand-side, so can update
  the variable to make right-hand-side the least upper
  bound of both side
Convergence on Infinite Lattice

“Bad” constraint:
\[(\alpha)\text{array} \leq \alpha \leq \text{unk}\] 
\[(\text{unk})\text{array} \leq \text{unk} \leq \text{unk}\] 
\[(\text{unk})\text{array} \leq \text{unk} \leq \text{unk} \leq \text{unk} \leq \text{unk} \ldots\]

Convergence on Infinite Lattice (cont.)

- All chains from any constant type to the top are finite
- Infinite updates with unknown types can be detected by setting a bound on the depth of structured types
Actors Manipulating Structured Types

- SequenceToArray
- ArrayToSequence
- ArrayAppend
- ArrayElement
- ArrayExtract
- ArrayLength
- RecordAssembler
- RecordDisassembler
- RecordUpdater

Using Monotonic Functions in Type Constraints

- Inequality solving algorithm admits:
  \[
  \text{const } \alpha f(\alpha) \leq \text{const } \alpha
  \]

- Inequalities \( f(\alpha) \leq \beta \) can express complex type constraints
Monotonic Function

- A function \( f: T \rightarrow T \) is monotonic if \( x \leq y \) implies \( f(x) \leq f(y) \)

AbsoluteValue Actor

- **Requirements**
  - Works for Int, Long, Fix, Double, Complex
  - Output type is the same as the input, unless input is Complex
  - Output type is Double when input is Complex

\[ f(\alpha) \leq \beta, \text{ where} \]

\[ f(\alpha) = \begin{cases} 
\text{Double} & \text{if } \alpha = \text{Complex} \\
\alpha & \text{otherwise} 
\end{cases} \]
RecordUpdater Actor

\[
f(\alpha, \beta_1, \beta_2) \leq \gamma
\]

<table>
<thead>
<tr>
<th>(\alpha)</th>
<th>(\beta_1)</th>
<th>(\beta_2)</th>
<th>(f(\alpha, \beta_1, \beta_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>unknown</td>
<td>Double</td>
<td>Int</td>
<td>unknown</td>
</tr>
<tr>
<td>(item: String, val: Int)</td>
<td>Double</td>
<td>Int</td>
<td>(item: String, val: Double, id: Int)</td>
</tr>
</tbody>
</table>

Conclusion

- Structured types increases the expressiveness of the modeling environment
- Monotonic function can describe complex type constraints