Treo: Textual Syntax of Reo

Kasper Dokter and Farhad Arbab
Context

Component-based **software** engineering

Construction of system/architecture (in Reo)
### Reo

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \rightarrow b$</td>
<td>A Sync channel accepts datum from its source end $a$, when its simultaneous offer of this datum at its sink end $b$ succeeds.</td>
</tr>
<tr>
<td>$a \rightarrow b$</td>
<td>A SyncDrain channel simultaneously accepts data from both its source ends $a$ and $b$ and loses the data.</td>
</tr>
<tr>
<td>$a \rightarrow b$</td>
<td>An empty FIFO accepts data from its source end $a$ and becomes a full FIFO. A full FIFO offers its stored data at its sink end $b$ and, when its offer succeeds, it becomes an empty FIFO again.</td>
</tr>
</tbody>
</table>

A Reo node accepts a datum from one of its coincident sink ends ($a$ or $c$), when its simultaneous offer to dispense a copy of this datum through every one of its coincident source ends ($b$ and $d$) succeeds.
Graphical editor (ECT)

No iterated definitions
No recursive definitions
No conditional construction
FOCAML

Used by Jongmans for a Reo compiler

Primitive components are constraint automata.

FOCAML

Used by Jongmans for a Reo compiler


Primitive components are constraint automata.
Why Treo

“users/devs usually enrich the input language for driving the code generator. Avoid the burden of changing the core grammar as this is very often overkill.”

developer reference BIP2-compiler
Why Treo

“users/devs usually enrich the input language for driving the code generator. **Avoid** the burden of **changing** the **core grammar** as this is very often **overkill.”**

*developer reference BIP2-compiler*

We need a stable core grammar.

*Treo* is a language for **compositional** construction of **systems** from user-defined primitives.
Structure of Treo code

```java
import syncdrain;
import sync;

fifo1(a?, b!) {
    empty -{a}, true-> full;
    full -{b}, true-> empty;
}

alternator2(a1, a2, b1) {
    sync(a1, b1)
    syncdrain(a1, a2)
    sync(a2, b2)
    fifo1(b2, b1)
}
```

Imports

Primitive component (in user-defined syntax)

Composite component

Writes to b1
User-defined primitives

myPrimitive1(a?,b!) { 
  b'(t) = x(t) * b(t)^2
  x'(t) = a(t)
}

myPrimitive2(a?,b!) { 
  empty -{a},true-> full;
  full -{b},true-> empty;
}

myPrimitive3(a?,b!) { 
  "com.example.MyJavaClass"
}

myPrimitive4(a?,b!) { ... }

Component type: systems of differential equations

Labeled transition systems

Source files

Generic component type T
User-defined primitives

Definition (informal):

A component type is a collection of composable component instances with interfaces

Examples:

- All systems of differential equations
- All Petri-nets (with suitable composition)
- All temporal formulas (TLA+)
- ...

Parameters and iteration

import syncdrain;
import sync;
import fifol;

alternator<k>(a[1:k], b[1]) {
    sync(a[1], b[1])
    { syncdrain(a[i-1], a[i])
      sync(a[i], b[i])
      fifol(b[i], b[i-1]) | i in [2..k] }
}
Enriched parameter passing

**Definition passing**

```java
import myDef;

pattern1<C>() {
    C(a,b)
}

main1() {
    pattern<myDef>()
    pattern<myDef>()
}
```

**Instance passing**

```java
import myDef;

pattern2<C>() {
    C
}

main2() {
    pattern<myInst>()
    pattern<myInst>()
    myInst = myDef(a,b)
}
```
Recursion

import fifo1;
import sync;

buffer<k>(a,b) {
    if (k > 0) {
        buffer<k-1>(a,x)
        fifo1(x,b)
    } else {
        sync(a,b)
    }
}

Recursion is syntactically allowed, but the current semantics does not yet support this feature.

The current implementation already supports bounded recursion.
Applications

We use Treo in ongoing work on a Reo compiler

\[ \text{alternator}^{<500>()} \text{ has } 1.6 \times 10^{150} \text{ states and compiles in 12 sec.} \]
Conclusion

- Treo = composition
- User-defined primitives → use your favorite type!
- Iteration, recursion, enriched parameter passing

Future work

- Semantics of recursive definitions & instance identities.
- Dynamic reconfiguration
- Implement BIP to Reo translator