Funcons
Executable Component-Based Semantics

L. Thomas van Binsbergen,
Neil Sculthorpe, Peter D. Mosses

Royal Holloway, University of London

22 March, 2015
Section 1

PLanCompS
Reusable Components: Funcons

Java    Java Core

Diagram of dependencies and relationships between Java and Java Core components.
Reusable Components: Funcons

Java  Java Core  C#  C# Core  C#

L. Thomas van Binsbergen, Neil Sculthorpe, Peter D. Mosses
Reusable Components: Funcons

<table>
<thead>
<tr>
<th>Java</th>
<th>Funcons</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://example.com/diagram.png" alt="Diagram" /></td>
<td><img src="https://example.com/diagram.png" alt="Diagram" /></td>
<td><img src="https://example.com/diagram.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

L. Thomas van Binsbergen, Neil Sculthorpe, Peter D. Mosses
The PLanCompS Approach

- Component based approach towards formal semantics.
- The components are highly reusable constructions.
- We call them fundamental constructions or \textit{funcons}.
- Each funcon has a formal definition in I-MSOS.
Figure: PLanCompS: generate interpreters from reusable specification.
Tool Support

Spoofax / Eclipse
- CBS is implemented in the Spoofax language workbench.
- IDE support, e.g. syntax-highlighting, declaration referencing.

Haskell
- Modular funcon definition (both manual and generated).
- Compositional interpreters (plug and play).
- Hackage: funcons-tools, gll.
Case Studies

- Caml Light case study completed (TAOSD2015).
- C# case study underway.
- Various small languages: imperative & logical programming.
- Website: http://plancomps.org
Section 2

CBS Funcon Compilation
Implicitly Modular Structural Operational Semantics.

I-MSOS transition relations:
- Context-free rewrites $X = Y$.
- Context-aware small steps $X \rightarrow Y$.

Semantic entities propagate contextual information.

Every funcon has ‘zero or more’ step and/or rewrite rules.

Funcon terms are values or computations.
Example - If-Then-Else

Funcon \texttt{if-then-else}(\_ : \texttt{booleans}, \_ : \Rightarrow T, \_ : \Rightarrow T) : \Rightarrow T

Rule \texttt{if-then-else}(\texttt{true}, X, \_) = X

Rule \texttt{if-then-else}(\texttt{false}, \_, Y) = Y
Example - Bound

Funcon $\text{bound}(\_ : \text{identifiers}) : \Rightarrow \text{values}$

Rule

$\frac{\text{lookup}(B, \rho) = V}{\text{environment}(\rho) \vdash \text{bound}(B) \rightarrow V}$
Funcons in Haskell

- Rules are implemented as sequences of monadic statements.
- Both transition relations have their own monad.
- Both monads propagate meta-information.
- Only the step-monad propagates semantic entities.
- An applicable rule is selected by backtracking.
\[
C_1 \ldots C_k \\
\frac{f(P)}{= T}
\]
\[ X : \text{booleans} \]

\[ f(P) = T \]
\[ Y \equiv true \]
\[ f(P) = T \]
Z = [1, X]
\[
\frac{f(P)}{T}
\]
Rewrite Rules

\[
\frac{C_1 \ldots C_k}{f(P) = T}
\]

\[
R = \text{do} \\
\text{let } env = \text{emptyEnv} \\
env \leftarrow \text{fsMatch fargs } P \text{ env} \\
env \leftarrow \text{sideCondition } C_1 \text{ env} \\
\ldots \\
env \leftarrow \text{sideCondition } C_k \text{ env} \\
\text{substitute } T \text{ env}
\]
Backtracking

- The statements of a rule can throw exceptions.
- Some exceptions indicate a rule is not applicable.
- Other exceptions indicate an internal error.
- A handler function backtracks between rules, until
  - A rule has been executed successfully (it was applicable).
  - A rule throws an internal error, which is then propagated.
CBS supports the definition of *semantics entities*. Each belonging to one of five entity classes:

- Inherited, e.g. *environment*
- Mutable, e.g. *store*
- Output, e.g. *standard-out*
- Input, e.g. *standard-in*
- Control, e.g. *thrown*

In a single rule multiple entities of the same or different classes can be used.

Each entity class implemented by a map, achieving modularity.
Inherited Entities

\[ \text{environment}(\gamma) \vdash f(P) \rightarrow T \]

\[ S = \text{do} \]
\[ \text{let } env = \text{emptyEnv} \]
\[ env \leftarrow \text{fsMatch fargs P env} \]
\[ env \leftarrow \text{getInhPatt "environment" } \gamma \text{ env} \]
\[ \ldots \]
\[ \text{substitute } T \text{ env} \]
Inherited Entities as Premises

\[ T \rightarrow P \]

\[ \cdots \]

\[ \text{env} \leftarrow \text{stepTerm} \ T \ P \ \text{env} \]

\[ \cdots \]
Inherited Entities as Premises

\[ \text{environment}(\gamma) \vdash T \rightarrow P \]

... 

\[ env \leftarrow \text{withinTerm} \text{ "environment" } \gamma \text{ env} \]
\[ (\text{stepTerm} T P env) \]

...
Mutable Entities

\[ \langle f(P), \text{store}(\sigma) \rangle \rightarrow \langle T, \text{store}(\sigma') \rangle \]

\[ S = \text{do} \]

\begin{verbatim}
  let env = emptyEnv
  env ← fsMatch fargs P env
  env ← getMutPatt "store" σ env
  ...
  putMutTerm "store" σ' env
  substitute T env
\end{verbatim}
Mutable Entities as Premises

\[ \langle T, \text{store}(\sigma') \rangle \rightarrow \langle P, \text{store}(\sigma) \rangle \]

\[ \ldots \]

\[ \text{putMutTerm} \quad \text{“store”} \quad \sigma' \quad \text{env} \]
\[ \text{env} \leftarrow \text{stepTerm} \quad T \quad P \quad \text{env} \]
\[ \text{getMutPatt} \quad \text{“store”} \quad \sigma \quad \text{env} \]

\[ \ldots \]
Pattern Matching

- CBS supports meta-variables like $X^*$, $Y^+$ and $Z$.
- Example of an ambiguous pattern: $X^*$, $Y^+$.
- Greedy longest-match is no solution: $X^*$, $true$. 
Pattern Matching Algorithm

- Assuming a matcher $M$ for single terms and patterns.
- We can then match $t_1, \ldots, t_n$ with $p_1, \ldots, p_m$.
- Starting with $(0, 0, emptyEnv) \in \mathcal{R}$, until $\mathcal{R} \equiv \emptyset$:
  - Pop $(i, j, env)$ from $\mathcal{R}$.
  - Return $env$ if $i \equiv n + 1$ and $j \equiv m + 1$.
  - Next iteration if $i < n + 1$ and $j \equiv m + 1$.
  - If $p_j$ is a simple pattern:
    - Add $(i + 1, j + 1, env') \in \mathcal{R}$ iff $env' = M(t_i, p_j)$.
  - Else $p_j$ is a sequence meta-variable: ... (next slide)
- Pattern mismatch if no $env$ was returned.
If \( p_j \) is a sequence meta-variable:

- Add \((k, j + 1, env') \in R\), with:
  - \( \forall i \leq k \leq n + 1 \) (if \( p_j \) is \( X^* \))
  - \( \forall i < k \leq n + 1 \) (if \( p_j \) is \( X^+ \))
  - \( \forall i \leq k \leq i + 1 \) (if \( p_j \) is \( X? \))
  - \( env' = [X^* \mapsto t_i, \ldots, t_{k-1}]env \).