A Compositional Typed Higher-Order Logic with Definitions

Ingmar Dasseville - Matthias Van Der Hallen - Bart Bogaerts - Gerda Janssens - Marc Denecker

International Conference on Logic Programming, 2016
Goal in KR:

- build expressive logics
- by integrating useful and expressive language constructs
- in a meaning preserving way
To add aggregate expressions to logic programming and ASP: many effort years, several PhD’s and many papers.
To add a nested cardinality aggregate $\text{Card}$ to classical logic:

- **New syntactical rule** in definition of term:
  - $\text{Card}(\{x : \varphi\})$ is a term if $\varphi$ is a formula

- **New semantical rule** in definition of term evaluation:
  - $(\text{Card}(\{x : \varphi\}))\models = \#(\{d \mid \mathcal{I}[x : d] \models \varphi\})$
To add a nested cardinality aggregate $\text{Card}$ to classical logic:

- **New syntactical rule in definition of term:**
  - $\text{Card}(\{x : \varphi\})$ is a term if $\varphi$ is a formula

- **New semantical rule in definition of term evaluation:**
  - $(\text{Card}(\{x : \varphi\}))^\mathcal{I} = \#(\{d \mid \mathcal{I}[x : d] \models \varphi\})$

We are ready.
Developing a compositional method to extend rule sets under well-founded and stable semantics with new language constructs.
1 – Timeline

**Last Year:** Adding templates to KR languages

**Result:** Framework for adding language constructs and building logics

**This Year:** Building a general logic including compositionality principles
Definition (Compositionality according to Frege)

The meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them.
The semantics for a logic $L$ and a language constructs $C$ must satisfy:

$$Sem_L(C(e_1, ..., e_n)) = Sem_C(Sem_L(e_1), ..., Sem_L(e_n))$$
What is $\text{Sem}_L(C(e_1, ..., e_n))$ mathematically?

- Logic expressions express “information”
- Infon: mathematical semantical object to express information
  - Function from structures to values
  - $= A$ quantum of information
  - Confer intensional objects (e.g., Montague)
What is $Sem_L(C(e_1,\ldots,e_n))$ mathematically?

- Logic expressions express “information”
- Infon: mathematical semantical object to express information
  - Function from structures to values
  - $=$ A quantum of information
  - Confer intensional objects (e.g., Montague)

Infon of $p \lor q$
Maps $\{p\}$ to True
Maps $\{}$ to False
What is $\text{Sem}_L(C(e_1,\ldots,e_n))$ mathematically?

- Logic expressions express “information”
- Infon: mathematical semantical object to express information
  - Function from structures to values
  - $=\,$ A quantum of information
  - Confer intensional objects (e.g., Montague)

<table>
<thead>
<tr>
<th>Infon of $p \lor q$</th>
<th>Infon of $c + 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps ${p}$ to $\text{True}$</td>
<td>Maps ${c = 5}$ to $8$</td>
</tr>
<tr>
<td>Maps ${}$ to $\text{False}$</td>
<td></td>
</tr>
</tbody>
</table>
**Syntax:** Extend the set of valid expressions
Syntax: Extend the set of valid expressions

Typing: Not all expressions within the grammar are sensible (e.g. 1+“hello”)
Syntax: Extend the set of valid expressions

Typing: Not all expressions within the grammar are sensible (e.g. 1+“hello”)

Semantics: What information corresponds to the expression?
A language expression $C$ is:

- Abstract syntax $C(e_1, ..., e_n)$
A language expression $C$ is:

- Abstract syntax $C(e_1, \ldots, e_n)$
- Typing function on types: $\text{Typ}_C(type_1, \ldots, type_n)$ such that

$$\text{Typ}_L(C(e_1, \ldots, e_n)) = \text{Typ}_C(\text{Typ}_L(e_1), \ldots, \text{Typ}_L(e_n))$$
A language expression $C$ is:

- Abstract syntax $C(e_1, \ldots, e_n)$
- Typing function on types: $Typ_C(type_1, \ldots, type_n)$ such that
  $$Typ_L(C(e_1, \ldots, e_n)) = Typ_C(Typ_L(e_1), \ldots, Typ_L(e_n))$$
- Semantic function on infons: $Sem_C(Infon_1, \ldots, Infon_n)$ such that
  $$Sem_L(C(e_1, \ldots, e_n)) = Sem_C(Sem_L(e_1), \ldots, Sem_L(e_n))$$
Simply typed lambda calculus

- Higher order types
- Lambda Abstractions

Definitions

- Higher order Rules
- Well-founded/stable semantics, lifted
Higher Order Definitions

{ 
∀ cur ∀ Move ∀ IsWon:
  win(cur, Move, IsWon) ← IsWon(cur) ∨
  ∃ nxt : Move(cur, nxt) ∧ lose(nxt, Move, IsWon).

∀ cur ∀ Move ∀ IsWon:
  lose(cur, Move, IsWon) ← ¬IsWon(cur) ∧
  ∀ nxt : Move(cur, nxt) ⇒ win(nxt, Move, IsWon).
}
• Meaning of a logical expression is an infon.
• Compositionality obtained using Frege’s principle.
• Integration of common logical and functional language constructs.
• Simplifying current and enabling new applications.
• But we need solvers!