Semantic Types (Sorts) in SNePS 3
Presentation to SNeRG
March 28, 2008

Stuart C. Shapiro
Department of Computer Science and Engineering
and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000

shapiro@cse.buffalo.edu
Sorted Logic

Introduce a hierarchy of sorts, $s_1, \ldots, s_n$.

A sort in logic is similar to a data type in programming.
Ergo sorts = semantic types.

Assign each individual constant a sort.
Assign each variable a sort.

Declare the sort of each argument position of each function symbol, and of each predicate symbol.

A functional term $f^n(t_1, \ldots, t_n)$, or an atomic formula, $P^n(t_1, \ldots, t_n)$
is syntactically valid only if the sort of $t_i$, for each $i$,
is the sort, or a subsort of the sort, declared for the $i^{th}$ argument position of $f^n$ or $P^n$. 

Page 2
Sorted Logic and Unification

If $E_1$ and $E_2$ are expressions of a sorted logic,

$\{t_1/v_1, \ldots, t_n/v_n\}$ is an mgu of $E_1$ and $E_2$

only if the sort of $t_i$ is a (not necessarily proper) subsort of $v_i$. 
Sorts Not in Object Language

Instead of
\[ \forall x [ \text{Person}(x) \Rightarrow (\text{Parent}(x) \Leftrightarrow \exists y (\text{Person}(y) \land \text{hasChild}(x, y)))] \]

hasChild(Betty, Tom)

and having to prove

\text{Person}(Betty) \text{ and } \text{Person}(Tom)

in order to prove that \text{Parent}(Betty)

Can declare

Terms of sort \text{person}: Betty, Tom, x, y
\text{Parent}'s argument of sort: \text{person}.
\text{hasChild}'s arguments of sort: \text{person}, \text{person}.

and

\[ \forall x [\text{Parent}(x) \Leftrightarrow \exists y \text{ hasChild}(x, y)] \]
Semantic Types (Sorts) in SNePS 3

Note: Every well-formed expression is a term.
So have individual constants, function symbols,
and functional terms.

Types are sorts:
(defineType type (supertype ...) ...)

Relations specify argument positions:
(defineRelation name :type type ...)

Caseframes specify functional terms:
(defineCaseframe type '(funSymbol argname ...) ...)

Optional:
(declareTerm term type)
Contextual Determination of Term Types

(assert '(PropFun arg1 ... argi ...))
If had
(defineCaseframe type
    ('PropFun arg1name ... arginame ...)
    ...
)
and
(defineRelation arginame :type argitype ...)
Then the type of argi must be argitype.
Second Specification of a Term’s Type

If term is already in the KB with type currenttype, and newtype is specified for it via context or declareTerm:

- if currenttype is a subtype of newtype, the type of term is left as is;
- elseif newtype is a subtype of currenttype, the semantic type of term is lowered to newtype;
- elseif currenttype and newtype have one greatest common subtype, the semantic type of term is changed to that type;
- elseif currenttype and newtype have several greatest common subtypes, the user is asked which one (s)he wants term to be, and term’s semantic type is changed to that type;
- else, an error is generated.
Demonstration

Defining some types, relations, and caseframes:

; (defineType Agent (Thing) "Individuals that have agency")
; (defineType Human (Agent) "Humans")
; (defineType Robot (Agent) "Robots")

; (defineRelation agents :type Agent
docstring "The fillers are asserted to be agents.")
; (defineRelation humans :type Human
docstring "The fillers are asserted to be humans.")
; (defineRelation robots :type Robot
docstring "The fillers are asserted to be robots.")
; (defineRelation objects :type Thing
docstring "The fillers are things that are being given properties.")
; (defineRelation properties :type Thing
docstring "The fillers are properties of things.")

; (defineCaseframe 'Proposition '('Agent agents)
docstring "[agents] is an agent.")
; (defineCaseframe 'Proposition '('Human humans)
docstring "[humans] is a human.")
; (defineCaseframe 'Proposition '('Robot robots)
docstring "[robots] is a robot.")
Steve is of type Human, therefore also Agent:

(assert '(Human Steve))

wft1!: (Human Steve)

(type-of (find-term 'Steve))

atom-Human

(assert '(Agent Steve))

wft2!: (Agent Steve)

(type-of (find-term 'Steve))

atom-Human
Cassie begins as Agent; lowered to Robot:

: (assert '(Agent Cassie))
wft3!: (Agent Cassie)
: (type-of (find-term 'Cassie))
atom-Agent

: (assert '(Robot Cassie))
wft4!: (Robot Cassie)
: (type-of (find-term 'Cassie))
atom-Robot
Human and Robot are incompatible, but can introduce Cyborg.

: (assert '(Robot Steve))
Error: The attempt to use Steve in a context that requires it to be of type 
Robot conflicts with its current type of Human.

: (defineType Cyborg (Human Robot) "Bionic people")

: (assert '(Robot Steve))
wft5!: (Robot Steve)
: (type-of (find-term 'Steve))
aton-Cyborg
An example of multiple most-general common subtypes:

```
(defineType Movie (Thing) "Movies")
(defineType Play (Thing) "Plays")
(defineType ActionMovie (Movie) "Action movies")
(defineType CostumeEpic (Movie) "Costume Epics")
(defineType MysteryPlay (Play) "Mystery plays")
(defineType HistoricalPlay (Play) "Historical drama-play")
(defineType HistoricalMysteryMovie (Movie) "Movie of a mystery play in historical costumes")
(defineType HistoricalActionMovie (Movie) "Action movie version of an historical play")

(defineRelation movies :type Movie
docstring "The fillers are movies.")
(defineRelation plays :type Play
docstring "The fillers are plays.")

(defineCaseframe 'Proposition '('Movie movies)
docstring "[movies] is a movie.")
(defineCaseframe 'Proposition '('Play plays)
docstring "[plays] is a play.")
```
An example of multiple most-general common subtypes:

: (assert '(Play "Murder in the forum"))
wft6!: (Play |Murder in the forum|)

: (assert '(Movie "Murder in the forum"))
Choose a type for |Murder in the forum|.
1. #<standard-class HistoricalActionMoviePlay>
2. #<standard-class HistoricalMysteryMoviePlay>
3. Cancel
2
wft7!: (Movie |Murder in the forum|)

(type-of (find-term "Murder in the forum"))
output:
atom-HistoricalMysteryMoviePlay
The user can declare a term to be of some type ahead of time.

: (declareTerm 'Macbeth 'HistoricalActionMoviePlay)
Macbeth

: (assert '(Play Macbeth))
: (assert '(Movie Macbeth))
: (type-of (find-term 'Macbeth))
atom-HistoricalActionMoviePlay
An Example of subtypes of Proposition.

: (clearkb t)
: (defineType ReifiedProposition (Proposition))
: (defineType Relation (Thing))
: (defineType MetaPredicate (Relation))

: (defineRelation propositionalargument :type ReifiedProposition)
: (defineRelation metapreds :type MetaPredicate)

: (defineCaseframe 'Proposition '(metapreds propositionalargument)
  :symbols '(Holds)
  :docstring "[propositionalargument] [metapreds] in the current situation.")
Using subtypes of Proposition.

: (setf AnBareBlocks (sneps:build '(Isa (setof A B) Block) 'Proposition))
wft1?: (Isa (setof B A) Block)

: (type-of AnBareBlocks)
categorization-Proposition

: (assert '(Holds ,AnBareBlocks))
wft2!: (Holds (Isa (setof B A) Block))

: (type-of AnBareBlocks)
categorization-ReifiedProposition
Demonstration page 10

ReifiedPropositionalFluent is a subtype of ReifiedProposition

: (defineType ReifiedPropositionalFluent (ReifiedProposition))
: (defineType Situation (Thing))
: (defineType BinaryRelation (Relation))

: (defineRelation propfluentargs :type ReifiedPropositionalFluent)
: (defineRelation sitargs :type Situation)
: (defineRelation rel :type Relation)
: (defineRelation arg1 :type Entity)
: (defineRelation arg2 :type Entity)

: (defineCaseframe 'Proposition '(metapreds propfluentargs sitargs)
  :symbols '(HoldsIn)
  :docstring "[propfluentargs] [metapreds] in [sitargs]"
: (defineCaseframe 'Proposition '(rel arg1 arg2)
  :symbols '(On)
  :docstring "[rel] holds between [arg1] and [arg2]"

Page 17
Using ReifiedPropositionalFluents

: (assert '(HoldsIn (On A B) S0))

:(list=terms :types t)
<atom-Relation>  On
<atom-MetaPredicate> HoldsIn
<atom-MetaPredicate> Holds
<atom-Situation>  S0
<atom-Category>   Block
<atom-Entity>     A
<atom-Entity>     B
<molecular-Proposition> wft2!: (Holds (Isa (setof B A) Block))
<molecular-Proposition> wft4!: (HoldsIn (On A B) S0)
<categorization-ReifiedProposition> wft1?: (Isa (setof B A) Block)
<molecular-ReifiedPropositionalFluent> wft3?: (On A B)
Negations get types from arguments.

: (type-of (print (assert '(not (On A B)))))
  wft5!: (not (On A B))
  negation-ReifiedPropositionalFluent

: (type-of (print (assert '(nor (On A B) (Isa (setof A B) Block)))))
  wft6!: (nor (On A B) (Isa (setof B A) Block))
  negation-ReifiedProposition
Some Other Sorted KR Systems

DECREASONER
SNARK
Description Logics
A Peek at Sorts in Decreasoner

sort sort1,...,sortn
sort Constant1,...,Constantn
function Funsymbol(sort1,...):termsort
predicate Predsymbol(sort1,...)

Variables written as sorti

A Peek at Sorts in SNARK

(declare-sort sort)
(declare-disjointsorts sort1 sort2)
(declare-subsorts sort subsort1 ...)
(declare-sort-partition sort subsort1 ...)
(declare-sort 'sort :iff '(or sort1 sort2))
(declare-constant-symbol symbol :sort sort)
(declare-function-symbol fsym arity
   :sort '(termsort argsort1 ...))
(declare-predicate-symbol predsym arity
   :sort '(boolean argsort1 ...))

Variables can appear as ?sorti

Mark E. Stickel, Richard J. Waldinger, & Vinay K. Chaudhri, A Guide to
KIF+C is said to tie sorts to object-level classes. SNARK, when using KIF+C, reasons in the domain of sorts instead of the axioms of the object-level classes when appropriate.

Description Logics use object-level classes as sorts.
SNePS 3 Infers Object-level Categorizations from Types

(list-terms :types t)
(atom-Relation) on
(atom-MetaPredicate) HoldsIn
(atom-MetaPredicate) Holds
(atom-Situation) S0
(atom-Category) Block
(atom-Entity) A
(atom-Entity) B

(molecular-Proposition) wft2!: (Holds (Isa (setof A B) Block))
(negation-ReifiedPropositionalFluent) wft5!: (not (On A B))
(molecular-Proposition) wft4!: (HoldsIn (On A B) S0)
(negation-ReifiedProposition) wft6!: (nor (Isa (setof A B) Block) (On A B))
(categorization-ReifiedProposition) wft1?: (Isa (setof A B) Block)
(molecular-ReifiedPropositionalFluent) wft3?: (On A B)

: (ask '(Isa On Relation))
wft7!: (Isa On Relation)

: (ask '(Isa (On A B) ReifiedPropositionalFluent))
wft8!: (Isa (On A B) ReifiedPropositionalFluent)

: (ask '(Isa (setof Holds HoldsIn) (setof MetaPredicate Relation)))
wft9!: (Isa (setof HoldsIn Holds) (setof MetaPredicate Relation))