MetaLLic: A Specification Language for Linear Logic Programs

Chris Martens

January 12, 2015

Abstract

1 Examples

FPTP Voting

```
forall c:cand.
  CTX|{hopeful c _, defeated c, elected c} |=
  ((exists n.hopeful c n) + !defeated c + !elected c)
```

Social Simulation

```
forall c:character.
  CTX|{at _, dead c} |=
    ((exists l. at c l) + !dead c)
  CTX|{anger c c, aff c c, att c c} |=
    1
  forall o:object.
    CTX|{has c o} |= has c o + 1
```

Blocks World

```
forall b:block.
  CTX|{on b _, on_table b, arm_holding b} |=
    (exists b'. on b b') + on_table b + arm_holding b
  CTX|{on _ b, clear b, arm_holding b} |=
    (exists b'. on b' b) + clear b + arm_holding b

CTX|{arm_holding _, arm_free} |=
  (exists b. arm_holding b) + arm_free
```

Graphs

No self-edges; no multi-edges; otherwise arbitrary directed graph:

```
forall n:node.
  CTX|{edge n _} |=
    (exists n'. edge n n') + 1
  CTX|{edge _ n} |=
    (exists n'. edge n' n) + 1
  CTX|{edge n n} |= 1
```

Undirected graphs:

```
forall n,n':node.
  CTX|{edge n n', edge n' n} |=
    (edge n n' * edge n' n) + 1
```

Linked Lists

use case for recursion

```
mu list(d1,d2) =
  d1=d2 +
  (exists v,d'. at d1 v d' * list(d',d2))
CTX|{at _ _ _} |= exists d1, d2. list(d1,d2)
```

Q: equality – `d1=d2`?

Binary Counter

dps version of ordered fwd-chaining binary increment.

```
CTX|{inc} |= inc + 1
CTX|{at _ _ _} |= exists d1,d2. list(d1,d2)
```

program:

```
bit : type.
b0 : bit.
b1 : bit.

dest : type.
eos : dest.

inc : dest -> type.

at : dest -> bit -> dest -> type.

inc0 : inc D * at D b0 D' -o {at D b1 D'}.
inc1 : inc D * at D b1 D' -o {at D b0 D' * inc D'}.
ince : inc eos -o {1}.
```

2 Rule Checking

To check that a rule preserves the invariant:

1. Assume an arbitrary context admitting the premise obeys the property.
2. Invert on the metaLLic rules that make the property obtain.
3. Use the inversion facts to derive that the conclusion obeys the property.

Example: blocks world pick up from block.

```
pickupb : on X Y * clear X * arm_free -o
          {arm_holding X * clear Y}.
```

Premise context: Δ, on X Y, clear X, armfree
Know that the invariant holds for all blocks, namely X and Y: