Rewriting Logic Semantics

Mark Hills
MSPLS Spring 2006
Overview

- Introduction to rewriting logic
- Prior work
- Limitations of the current model
- Current focus and progress
- Future work
- Related work

Please jump in with comments!
Sorts and Signatures

- A Sort is essentially a Type
- Signatures can be unsorted, multi-sorted, or order-sorted
- Signatures contain at least a set of (uninterpreted) function symbols
- Multi-sorted signatures index this by sorts and contain the set of sorts
- Order-sorted signatures contain an ordering relation on sorts as well
Equational Logic

- An equational *theory* is a pair \((\Sigma, E)\), with \(\Sigma\) a signature and \(E\) a set of *equations*.
- Equations are of the form \(l = r\).
- Equations partition the set of terms formed from the signature into equivalence classes made up of provably equal terms.
A reduction system

- Treating equalities $l = r$ as reduction rules $l \rightarrow r$ yields a reduction system
- Languages are formulated as equational theories, programs are terms in the term algebra, reduction in an appropriate algebra = execution
- Programs reduced to canonical form – either termination in a final state, termination in a stuck state, or non-termination
Rewriting logic

- Equational logic can model non-concurrent computations
- Rewriting logic can be used to model concurrent computations -- threads, etc.
- Rewriting logic theories are triples $(\Sigma, E, R)$, where $R$ is a set of rewrite rules
- These rules allow transitions between equivalence classes of terms, $[t] \rightarrow [t']$, representing non-equivalent execution sequences
Prior work: overview

- Process calculi
  - CCS [Verdejo and Marti-Oliet, 2000]
  - $\pi$-Calculus [Thati, Sen and Marti-Oliet, 2002]
- Executable semantics
  - MSOS [Braga, 2001], [Chalub, 2005]
- Language design and definition
  - PLAN [Stehr and Talcott, 2002]
  - JavaFAN [Farzan, Meseguer and Rosu, 2004]
Prior work: directly related

- Pedagogical
  - UIUC Programming Language Design course (CS322/CS422), Special Topics course (CS497), Semantics (CS522) [Rosu]

- Language definition
  - Language subsets for class projects (CS522)
  - Scheme [d’Amorim and Rosu, 2004]
  - Beta [Hills, Aktemur and Rosu, 2005]
  - Java [Farzan, Chen, Meseguer, and Rosu, 2004]
Strengths of the current model

- Rewriting provides a simple model
- Fast enough for instructional use and prototyping
- Self-contained – no external tools needed
- Natural support for concurrency
- Good tool support (model checker, reachable state search, theorem prover, etc)
Some weaknesses

- Parsing is challenging for real languages
- Definitions not modular w.r.t. state changes for unrelated features
- Need to repeat unchanged portions of terms: unwieldy and error-prone
Modularity: example

- Before adding threads...

\[
\text{eq control}(k(\text{val}(V) \rightarrow \text{lassign}(L) \rightarrow K) \text{ CS}) \text{ mem}(\text{Mem} \ [L,V']) = \\
\text{control}(k(K) \text{ CS}) \text{ mem}(\text{Mem} \ [L,V]) .
\]

- And after...

\[
\text{rule} \ t(\text{control}(k(\text{val}(V) \rightarrow \text{lassign}(L) \rightarrow K) \text{ CS}) \text{ TS}) \text{ mem}(\text{Mem} \ [L,V']) \Rightarrow \\
\text{t}(\text{control}(k(K) \text{ CS}) \text{ TS}) \text{ mem}(\text{Mem} \ [L,V]) .
\]

- Nothing changed in this rule, but we still need to change it because of another feature
Repetition: example

- The current class (Xc) and class set don’t change, but we need to repeat them anyway

```plaintext
ceq control(k:invoke(Xm,Vl) \rightarrow K) CS cclass(Xc)
    cset(cls(cname(Xc) mthds(mthd(mname(Xm) mparams(Xs) mdecls(Xs’) mbody(Km)) Ms) CI) Cs) =
control(k:val(Vl) \rightarrow bind(Xs) \rightarrow bind(Xs’) \rightarrow Km \rightarrow K) CS cclass(Xc)
    cset(cls(cname(Xc) mthds(mthd(mname(Xm) mparams(Xs) mdecls(Xs’) mbody(Km)) Ms) CI) Cs)
if len(Vl) == len(Xs).
```
Current focus

- Define a rewriting notation specifically for programming languages: K notation
- Define a rewriting framework for languages: K framework
- Define paradigmatic languages:
  - SILF (imperative/procedural) [Hills, Serbanuta and Rosu, 2005]
  - KOOL (object-oriented) [Chen, Hills and Rosu, 2006]
  - FUN (functional) [Rosu, 2005]
- Define existing languages: Beta, Java, Scheme
K notation

- K is rewriting logic with PL-specific syntactic sugar
- Conciseness enhanced by sort inference
- Modularity enhanced by context transformers
- Syntax for list and set matching

\[ k(\langle V \mapsto L \rangle \text{mem} \langle (L, \frac{V'}{V}) \rangle) \]
Example: exceptions

- **Try/Catch**

\[
\begin{align*}
&(k(\frac{\text{try } S \text{ catch } X \text{ end}}{S \sim \text{popEStack}} \sim K) \text{ estack}(\frac{(\text{Ctrl}, \text{Env}, O, C, \text{bind(X)} \sim S' \sim \text{Env } \sim K)}{\text{Ctrl:CtrlState}}) \\
&\quad \text{env(Env) obj(O) class(C)}
\end{align*}
\]

- **Normal execution & throw**

\[
\begin{align*}
&(k(\frac{\text{popEStack}}{\text{}}) \text{ estack}(\frac{\text{}}{\text{}}) \\
&(k(V \sim \frac{\text{throw }}{K} \sim \text{}) \text{ estack}(\frac{(\text{Ctrl}, \text{Env}, O, C, K)}{\text{Ctrl}}) \sim \text{CtrlState}) \text{ env}(\frac{\text{ }}{\text{Env}}) \text{ obj}(\frac{\text{ }}{O}) \text{ class}(\frac{\text{ }}{C})
\end{align*}
\]
K framework

- Methodology for language definition
  - Explicit representation of control
  - Matching across inferred context boundaries
  - Stacks used to quickly recover prior state
  - Organization/nesting of state used to group related information

It isn’t necessary to use the K framework with the K methodology, but it is definitely cleaner
Language definition

■ Define example languages
  ■ Provides good examples for those interested in using K
  ■ Enhances confidence that technique is applicable to a broad number of languages

■ Define real languages
  ■ Again, increases confidence, plus provides unusual cases
  ■ Gives formal definitions that can then be used for language design and analysis

■ Both provide feedback as K develops
Future work

- Define additional languages
- Define new language features – after all, this is a platform for language design and experimentation
- Complete initial work on K to C and K to Rewrite Engine (Maude, ASF) translators
- Repository of language features
Future work?

- Support for working with theorem provers (Isabelle, Coq, etc)?
- Methods to enhance this to better support analysis?
- Nice GUI?
- Formalize mappings to other models of semantics?
Related Work

Semantics

- SOS [Plotkin, 1981]
- MSOS [Mosses, 2002][Mosses, 2004]
- Natural Semantics [Kahn, 1987]
- Denotational [Scott and Strachey, 1971][Schmidt, 1986]
- Monadc (denotational) [Moggi, 1989]
- Reduction [Felleisen and Hieb, 1992]
- Equational [Goguen, Thatcher, Wagner and Wright, 1977]
- Plus many, many more…
Related Work

- Executable semantics
  - Natural Semantics/Prolog [Clement, Despeyroux, Despeyroux, Hascoet and Kahn, 1985] [Borras, Clement, Despeyroux, Incerpi, Kahn, Lang and Pascual, 1988]
  - Many Semantics/Prolog [Slonneger and Kurtz, 1995]
  - Reduction Semantics/Scheme [Matthews, Findler, Flatt and Felleisen, 2004]
  - Rewriting-based [Goguen and Malcolm, 1996][van den Brand, Heering, Klint and Olivier, 2002]
  - Again, many more…
Questions?

Any questions or comments?