Program Representation for General Intelligence

The Reduct Toolkit for Program Normalization

Nil Geisweiller

Novamente LLC

Xiamen University
AGI Summer School 2009

Outline

Introduction

- Program learning is a very useful skill
- But program space is complex
- Understanding and exploiting its properties

$\Rightarrow$ Reducing programs in normal form

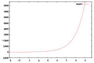
Outline

Program learning is HARD!

Program learning is HARD!

Program space is:

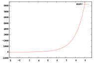
- Vast $\Rightarrow$ grows exponentially with program size



Program learning is HARD!

Program space is:

- Vast $\Rightarrow$ grows exponentially with program size

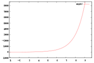


- Chaotic $\Rightarrow$ small syntactic variations can lead to huge semantic variations

Program learning is HARD!

Program space is:

- Vast $\Rightarrow$ grows exponentially with program size



- Chaotic $\Rightarrow$ small syntactic variations can lead to huge semantic variations

- Over-represented $\Rightarrow$ many programs with same semantics

$$x \wedge y \equiv x \wedge y \wedge x \equiv \neg(\neg(x) \vee \neg(y)) \equiv \dots$$

less diversity packed into the program space

Over-representation

Different syntax, same semantics

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Over-representation

Different syntax, same semantics

x	y	$x \wedge y$	$\neg(\neg y \vee \neg x)$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Over-representation

Different syntax, same semantics

x	y	$x \wedge y$	$\neg(\neg y \vee \neg x)$	$x \wedge x \wedge y$
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	1	1	1

Over-representation

Different syntax, same semantics

x	y	$x \wedge y$	$\neg(\neg y \vee \neg x)$	$x \wedge x \wedge y$	$x \wedge y \wedge (\neg z \vee z)$
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	1	1	1	1

Over-representation

Different syntax, same semantics

x	y	$x \wedge y$	$\neg(\neg y \vee \neg x)$	$x \wedge x \wedge y$	$x \wedge y \wedge (\neg z \vee z)$	...
0	0	0	0	0	0	...
0	1	0	0	0	0	...
1	0	0	0	0	0	...
1	1	1	1	1	1	...

Syntactic vs Semantic De-correlation

$$f_1 = x \wedge (y \vee z)$$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$f_2 = \neg(x \wedge (y \vee z))$$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2:

Syntactic vs Semantic De-correlation

$$f_1 = x \wedge (y \vee z)$$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$f_2 = \neg(x \wedge (y \vee z))$$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2:

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 1

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 2

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 3

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 4

Syntactic vs Semantic De-correlation

$$f_1 = x \wedge (y \vee z)$$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$f_2 = \neg(x \wedge (y \vee z))$$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 5

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 6

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 7

Syntactic vs Semantic De-correlation

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg(x \wedge (y \vee z))$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 8

Syntactic vs Semantic De-correlation

$g_1 = false$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Syntactic vs Semantic De-correlation

$$g_1 = \mathit{false}$$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$$g_2 = x \wedge y \wedge z$$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance:

Syntactic vs Semantic De-correlation

$g_1 = \textit{false}$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 1

Syntactic vs Semantic De-correlation

$g_1 = \textit{false}$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 2

Syntactic vs Semantic De-correlation

$g_1 = \mathit{false}$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 3

Syntactic vs Semantic De-correlation

$g_1 = \mathit{false}$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 4

Syntactic vs Semantic De-correlation

$g_1 = \mathit{false}$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 5

Syntactic vs Semantic De-correlation

$g_1 = false$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 6

Syntactic vs Semantic De-correlation

$g_1 = false$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 6
- Semantic distance:

Syntactic vs Semantic De-correlation

$g_1 = false$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 6
- Semantic distance: 1

Syntactic vs Semantic De-correlation

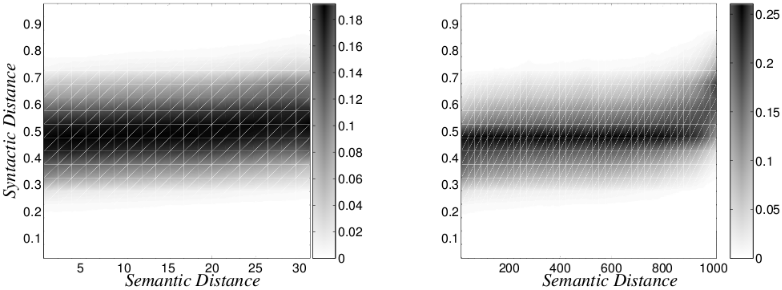


Figure: Syntactic vs semantic distance for random formulae with arities five (left) and ten (right). *Extracted from Moshe Looks PhD thesis.*

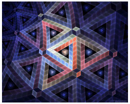
Outline

Programs in Normal Forms

1. unique representation (possibly the shortest)

$$x \wedge y \cancel{\equiv x \wedge y \wedge x \equiv \neg(\neg(x) \vee \neg(y)) \equiv \ldots}$$

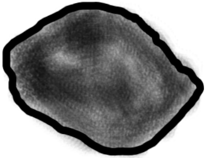
2. interesting properties $\Rightarrow$ program space more regular



Outline

Avoiding over-representation

Before reduction:



Program space

$$x \wedge y \equiv x \wedge y \wedge x \equiv \neg(\neg(x) \vee \neg(y)) \equiv \ldots$$
$$x \vee y \equiv x \vee y \vee x \equiv \neg(\neg(x) \wedge \neg(y)) \equiv \ldots$$
$$x \wedge y \wedge z \equiv x \wedge y \wedge z \wedge x \wedge y \wedge z \equiv \ldots$$
$$x \wedge y \wedge \neg(z) \equiv x \wedge y \wedge \neg(z) \wedge x \wedge y \wedge \neg(z) \equiv \ldots$$
$$\ldots$$

Avoiding over-representation

After reduction:

Program space

$$x \wedge y \cancel{\equiv x \wedge y \wedge x \equiv \neg(\neg(x) \vee \neg(y)) \equiv \ldots}$$
$$x \vee y \cancel{\equiv x \vee y \vee x \equiv \neg(\neg(x) \wedge \neg(y)) \equiv \ldots}$$
$$x \wedge y \wedge z \cancel{\equiv x \wedge y \wedge z \wedge x \wedge y \wedge z \equiv \ldots}$$
$$x \wedge y \wedge \neg(z) \cancel{\equiv x \wedge y \wedge \neg(z) \wedge x \wedge y \wedge \neg(z) \equiv \ldots}$$
$$\ldots$$

Increase Syntactic vs Semantic Correlation

Before reduction

$$f_1 = x \wedge (y \vee z) \qquad f_2 = \neg(x \wedge (y \vee z))$$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

After reduction

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg x \vee (\neg y \wedge \neg z)$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: ?
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

After reduction

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg x \vee (\neg y \wedge \neg z)$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 1
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

After reduction

$f_1 = x \wedge (y \vee z)$

$f_2 = \neg x \vee (\neg y \wedge \neg z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 2
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

After reduction

$f_1 = x \wedge (y \vee z)$

$f_2 = \neg x \vee (\neg y \wedge \neg z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 3
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

After reduction

$f_1 = x \wedge (y \vee z)$

$f_2 = \neg x \vee (\neg y \wedge \neg z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 4
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

After reduction

$f_1 = x \wedge (y \vee z)$

x	y	z	f_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f_2 = \neg x \vee (\neg y \wedge \neg z)$

x	y	z	f_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

- Syntactic distance between f_1 and f_2: 5
- Semantic distance between f_1 and f_2: 8

Increase Syntactic vs Semantic Correlation

Before and after reduction

$g_1 = \mathit{false}$

x	y	z	g_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$g_2 = x \wedge y \wedge z$

x	y	z	g_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

- Syntactic distance: 6
- Semantic distance: 1

Increase Syntactic vs Semantic Correlation

Before reduction

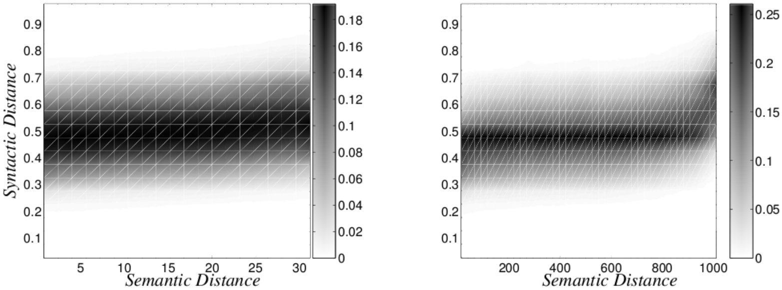


Figure: Syntactic vs semantic distance for random formulae with arities five (left) and ten (right). *Extracted from Moshe Looks PhD thesis.*

Increase Syntactic vs Semantic Correlation

After reduction

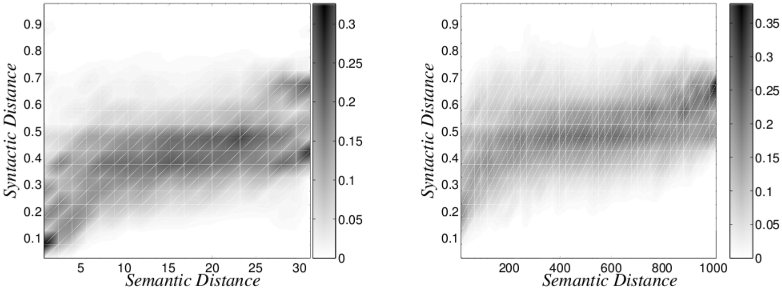
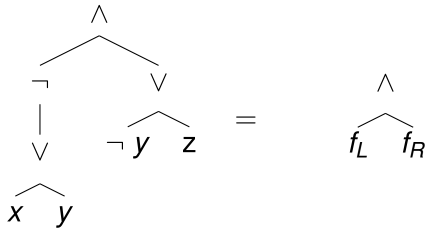


Figure: Syntactic vs semantic distance for random formulae in normal form with arities five (left) and ten (right). *Extracted from Moshe Looks PhD thesis.*

Improve Structure, interpretable, hierarchy

Before reduction:

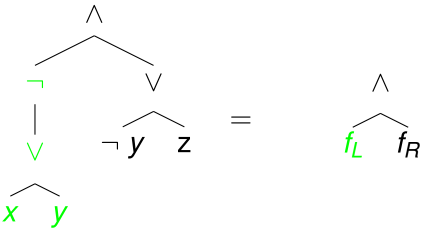
$$\neg(x \vee y) \wedge (\neg(y) \vee z) \quad =$$



Improve Structure, interpretable, hierarchy

Before reduction:

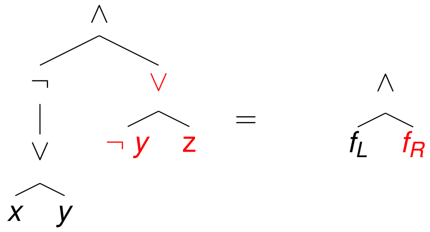
$$\neg(x \vee y) \wedge (\neg(y) \vee z) =$$



Improve Structure, interpretable, hierarchy

Before reduction:

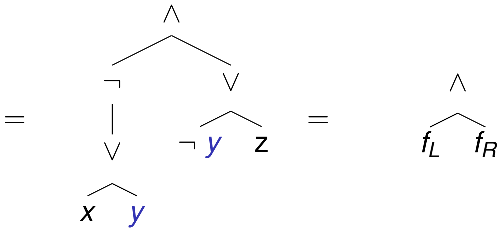
$$\neg(x \vee y) \wedge (\neg(y) \vee z) \quad = $$



Improve Structure, interpretable, hierarchy

Before reduction: f_L and f_R are not independent.

$$\neg(x \vee y) \wedge (\neg(y) \vee z)$$



Improve Structure, interpretable, hierarchy

After reduction: g_L and g_R are independent.

$$\neg x \wedge \neg y \quad = \quad \begin{array}{c} \wedge \\ \diagup \quad \diagdown \\ \neg x \qquad \neg y \end{array} \quad = \quad \begin{array}{c} \wedge \\ \diagup \quad \diagdown \\ g_L \qquad g_R \end{array}$$

Improve Structure, interpretable, hierarchy

After reduction: g_L and g_R are independent.

$$\neg x \wedge \neg y \quad = \quad \begin{array}{c} \wedge \\ \diagup \quad \diagdown \\ \neg x \qquad \neg y \end{array} \quad = \quad \begin{array}{c} \wedge \\ \diagup \quad \diagdown \\ g_L \qquad g_R \end{array}$$

Benefice

- Easier to understand
- Better hierarchical structure

To sum up

Reduction allows us to:

1. Avoid over-representation
2. Increase syntactic vs semantic correlation
3. Improve structure
4. Smaller programs take less memory and usually run faster

Conclusion

Reduction is good!

But beware!

Reduction to normal form is:

- undecidable for recursive functions 

But beware!

Reduction to normal form is:

- undecidable for recursive functions 
- undecidable for primitive recursive functions 

But beware!

Reduction to normal form is:

- undecidable for recursive functions 
- undecidable for primitive recursive functions 
- NP-hard for Boolean functions 

But beware!

Reduction to normal form is:

- undecidable for recursive functions 

- undecidable for primitive recursive functions 

- NP-hard for Boolean functions 

But beware!

Reduction to normal form is:

- undecidable for recursive functions 

- undecidable for primitive recursive functions 

- NP-hard for Boolean functions 

In practice

- incomplete reduction methods

But beware!

Reduction to normal form is:

- undecidable for recursive functions 
- undecidable for primitive recursive functions 
- NP-hard for Boolean functions 

In practice

- incomplete reduction methods
- rely on heuristics

But beware!

Reduction to normal form is:

- undecidable for recursive functions 
- undecidable for primitive recursive functions 
- NP-hard for Boolean functions 

In practice

- incomplete reduction methods
- rely on heuristics
- trade-off between efficiency and completeness

Outline

The Reduct Toolkit

- Part of the OpenCog framework

The Reduct Toolkit

- Part of the OpenCog framework
- Coded in C++

The Reduct Toolkit

- Part of the OpenCog framework
- Coded in C++
- Handles the following domains

The Reduct Toolkit

- Part of the OpenCog framework
- Coded in C++
- Handles the following domains
    - Boolean

The Reduct Toolkit

- Part of the OpenCog framework
- Coded in C++
- Handles the following domains
  - Boolean
  - Continuous

The Reduct Toolkit

- Part of the OpenCog framework
- Coded in C++
- Handles the following domains
  - Boolean
  - Continuous
  - numerico-boolean

The Reduct Toolkit

- Part of the OpenCog framework
- Coded in C++
- Handles the following domains
  - Boolean
  - Continuous
  - numerico-boolean
  - actions perceptions to control agent in virtual world

Outline

Reduct Operates on Combo

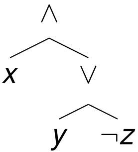
Combo is:

- Programmatic language of MOSES

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



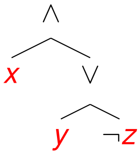
$\Leftrightarrow$

```
and(#1 or(#2 not(#3)))
```

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



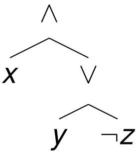
$\Leftrightarrow$

```
and(#1 or(#2 not(#3)))
```

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



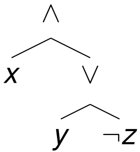
$\Leftrightarrow$ `and(#1 or(#2 not(#3)))`

- Boolean: `and`, `or`, `not`, `true`, `false`

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



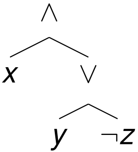
$\Leftrightarrow$ `and(#1 or(#2 not(#3)))`

- Boolean: `and`, `or`, `not`, `true`, `false`
- Continuous: $+$, $*$, $/$, `exp`, `log`, `sin`

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



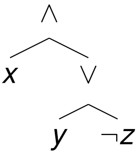
$\Leftrightarrow$ `and(#1 or(#2 not(#3)))`

- Boolean: `and`, `or`, `not`, `true`, `false`
- Continuous: `+`, `*`, `/`, `exp`, `log`, `sin`
- Numerico-Boolean: `0<`, `impulse`, `contin_if`

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



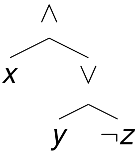
$\Leftrightarrow$ `and(#1 or(#2 not(#3)))`

- Boolean: `and`, `or`, `not`, `true`, `false`
- Continuous: `+`, `*`, `/`, `exp`, `log`, `sin`
- Numerico-Boolean: `0<`, `impulse`, `contin_if`
- Action: `and_seq`, `action_if`, `action_while`, `...`

Reduct Operates on Combo

Combo is:

- Programmatic language of MOSES
- Syntactic tree in prefix notation



$\Leftrightarrow$ `and(#1 or(#2 not(#3)))`

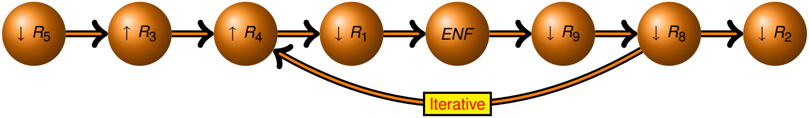
- Boolean: `and`, `or`, `not`, `true`, `false`
- Continuous: `+`, `*`, `/`, `exp`, `log`, `sin`
- Numerico-Boolean: `0<`, `impulse`, `contin_if`
- Action: `and_seq`, `action_if`, `action_while`, ...
- Higher order: *procedure_name*(*arity*) $:=$ *body*

Outline

Reduction

How it works

Apply series of reduction rules according to a given strategy.



Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: `and(`X`)` $\rightarrow X$

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: `and(`X`)` $\rightarrow X$
- R_5: `not(not(`X`))` $\rightarrow X$

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: `and(`X`)` $\rightarrow X$
- R_5: `not(not(`X`))` $\rightarrow X$
- R_8: `or(`X `true)` $\rightarrow$ `true`

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: `and(`X`)` $\rightarrow X$
- R_5: `not(not(`X`))` $\rightarrow X$
- R_8: `or(`X `true)` $\rightarrow$ `true`

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: $\texttt{and(}X\texttt{)} \rightarrow X$
- R_5: $\texttt{not(not(}X\texttt{))} \rightarrow X$
- R_8: $\texttt{or(}X\ \texttt{true)} \rightarrow \texttt{true}$

In the continuous domain:

- R_{13}: $X/Z+Y/Z \rightarrow (X+Y)/Z$

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: $\texttt{and(}X\texttt{)} \rightarrow X$
- R_5: $\texttt{not(not(}X\texttt{))} \rightarrow X$
- R_8: $\texttt{or(}X\ \texttt{true)} \rightarrow \texttt{true}$

In the continuous domain:

- R_{13}: $X\texttt{/}Z\texttt{+}Y\texttt{/}Z \rightarrow \texttt{(}X\texttt{+}Y\texttt{)/}Z$
- R_{21}: $\texttt{exp(}X\texttt{)}\star\texttt{exp(}Y\texttt{)} \rightarrow \texttt{exp(}X\texttt{+}Y\texttt{)}$

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: $\texttt{and(}X\texttt{)} \rightarrow X$
- R_5: $\texttt{not(not(}X\texttt{))} \rightarrow X$
- R_8: $\texttt{or(}X\texttt{ true)} \rightarrow \texttt{true}$

In the continuous domain:

- R_{13}: $X\texttt{/}Z\texttt{+}Y\texttt{/}Z \rightarrow \texttt{(}X\texttt{+}Y\texttt{)/}Z$
- R_{21}: $\texttt{exp(}X\texttt{)}\ast\texttt{exp(}Y\texttt{)} \rightarrow \texttt{exp(}X\texttt{+}Y\texttt{)}$

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: `and(`X`)` $\rightarrow X$
- R_5: `not(not(`X`))` $\rightarrow X$
- R_8: `or(`X `true)` $\rightarrow$ `true`

In the continuous domain:

- R_{13}: $X/Z+Y/Z \rightarrow (X+Y)/Z$
- R_{21}: `exp(`X`)*exp(`Y`)` $\rightarrow$ `exp(`$X+Y$`)`

In the action domain:

- R_{43}: `action_if(`B X Y`)` $\rightarrow X$ if B is reducible to `true`

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: and(X) $\rightarrow X$
- R_5: not(not(X)) $\rightarrow X$
- R_8: or(X true) $\rightarrow$ true

In the continuous domain:

- R_{13}: $X/Z+Y/Z \rightarrow (X+Y)/Z$
- R_{21}: exp(X)*exp(Y) $\rightarrow$ exp(X+Y)

In the action domain:

- R_{43}: action_if(B X Y) $\rightarrow X$ if B is reducible to true
- R_{51}: action_while(X ... X) $\rightarrow$ action_while(X)

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: and(X) $\rightarrow X$
- R_5: not(not(X)) $\rightarrow X$
- R_8: or(X true) $\rightarrow$ true

In the continuous domain:

- R_{13}: $X/Z+Y/Z \rightarrow (X+Y)/Z$
- R_{21}: exp(X)*exp(Y) $\rightarrow$ exp(X+Y)

In the action domain:

- R_{43}: action_if(B X Y) $\rightarrow X$ if B is reducible to true
- R_{51}: action_while(X ... X) $\rightarrow$ action_while(X)

Reduct Rules

Example of reduction rules in the Boolean domain:

- R_4: `and(`X`)` $\rightarrow X$
- R_5: `not(not(`X`))` $\rightarrow X$
- R_8: `or(`X `true)` $\rightarrow$ `true`

In the continuous domain:

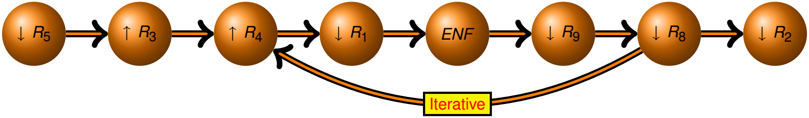
- R_{13}: X`/`Z`+`Y`/`Z $\rightarrow$ `(`X`+`Y`)/`Z
- R_{21}: `exp(`X`)`$\star$`exp(`Y`)` $\rightarrow$ `exp(`X`+`Y`)`

In the action domain:

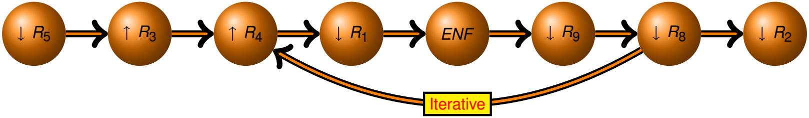
- R_{43}: `action_if(`B X Y`)` $\rightarrow X$ if B is reducible to `true`
- R_{51}: `action_while(`X `...` X`)` $\rightarrow$ `action_while(`X`)`

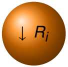
about 85 rules in total...

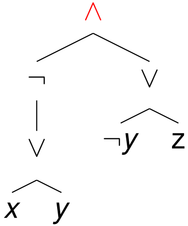
Combining the reduction rules



Combining the reduction rules

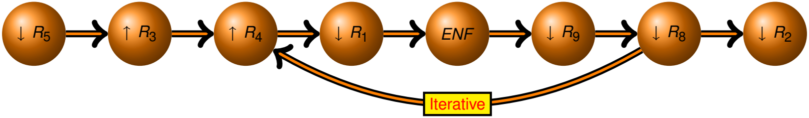


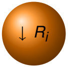


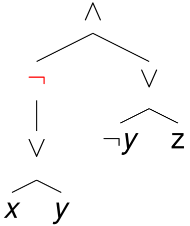


Apply R_i from the root down to the leaves.

Combining the reduction rules

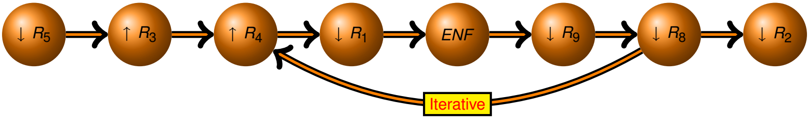


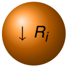


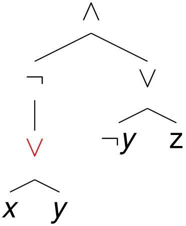


Apply R_i from the root down to the leaves.

Combining the reduction rules

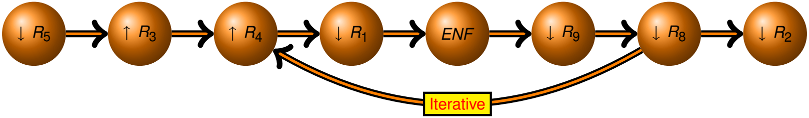


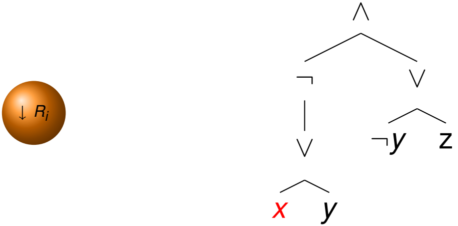




Apply R_i from the root down to the leaves.

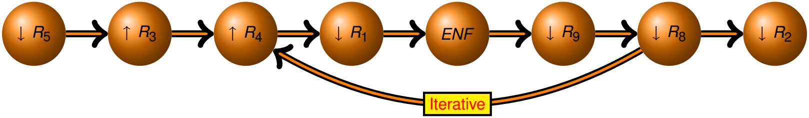
Combining the reduction rules

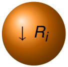


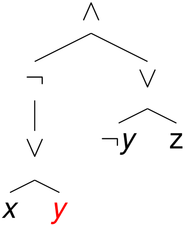


Apply R_i from the root down to the leaves.

Combining the reduction rules

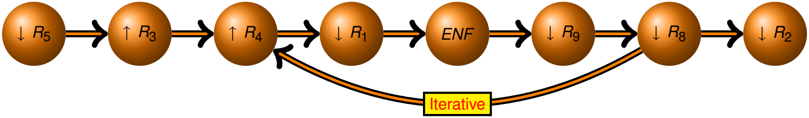


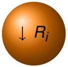


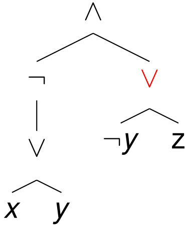


Apply R_i from the root down to the leaves.

Combining the reduction rules

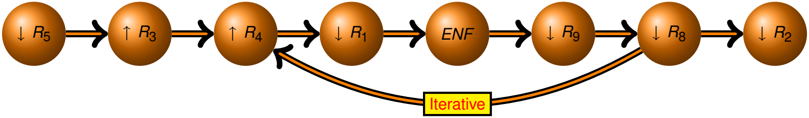


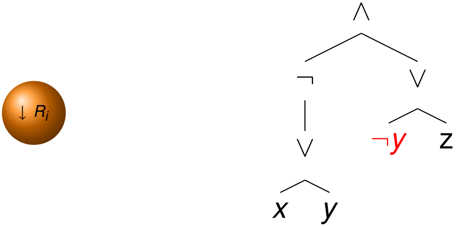




Apply R_i from the root down to the leaves.

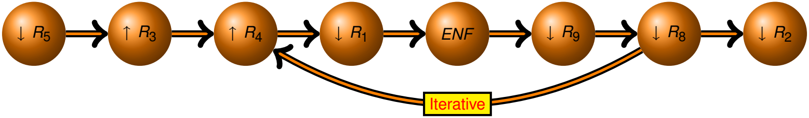
Combining the reduction rules

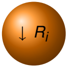


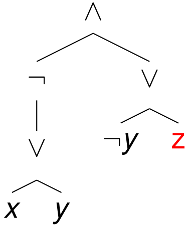


Apply R_i from the root down to the leaves.

Combining the reduction rules

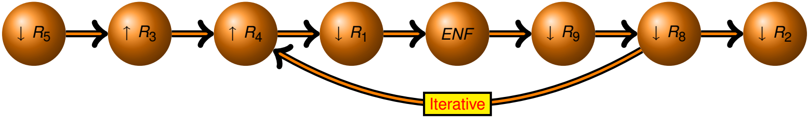


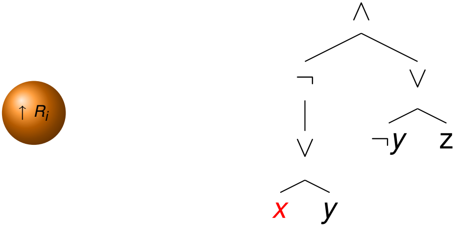




Apply R_i from the root down to the leaves.

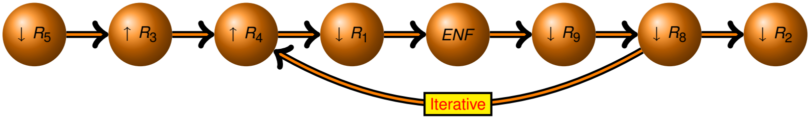
Combining the reduction rules

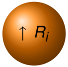


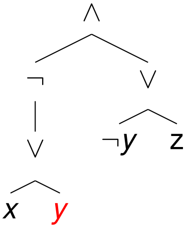


Apply R_i from the leaves up to the root.

Combining the reduction rules

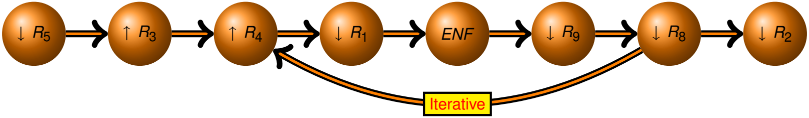


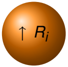


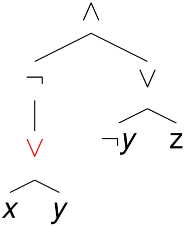


Apply R_i from the leaves up to the root.

Combining the reduction rules

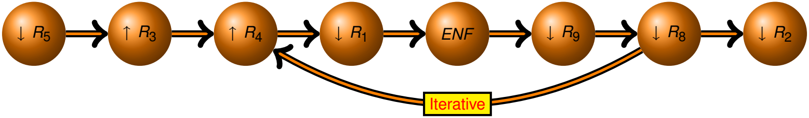


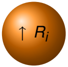


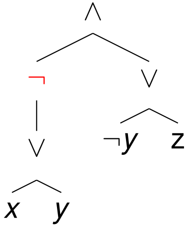


Apply R_i from the leaves up to the root.

Combining the reduction rules

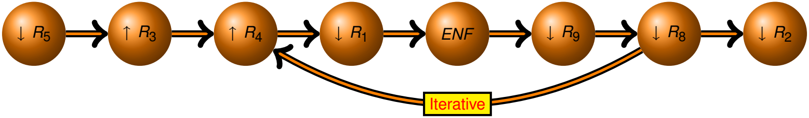


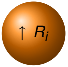


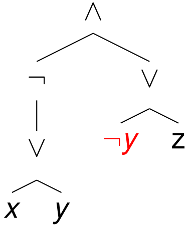


Apply R_i from the leaves up to the root.

Combining the reduction rules

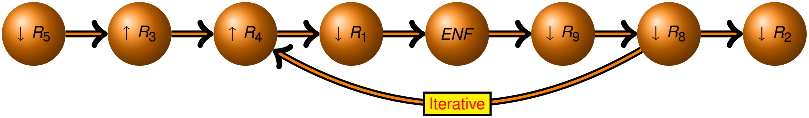


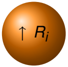


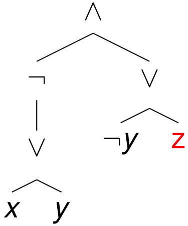


Apply R_i from the leaves up to the root.

Combining the reduction rules

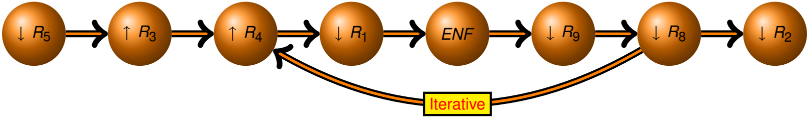


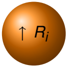


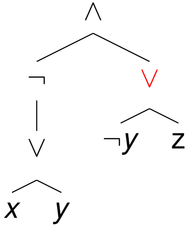


Apply R_i from the leaves up to the root.

Combining the reduction rules

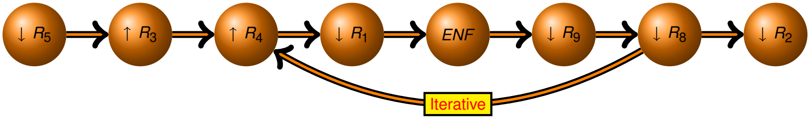


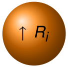


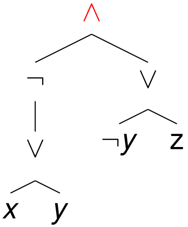


Apply R_i from the leaves up to the root.

Combining the reduction rules

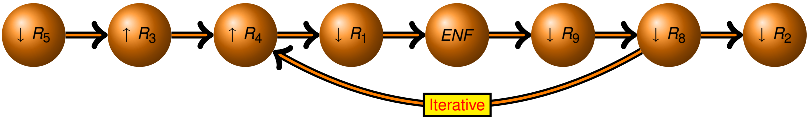


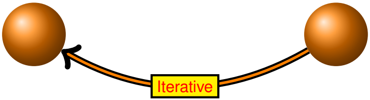




Apply R_i from the leaves up to the root.

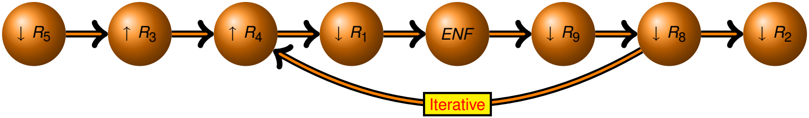
Combining the reduction rules

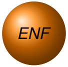




Repeat the strategy wrapped by `Iterative` until the combo tree no longer changes.

Combining the reduction rules

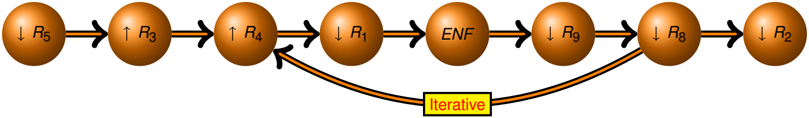


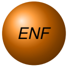


ENF: Elegant Normal Form (Holman, '90). A strategy onto itself.

- Efficient

Combining the reduction rules

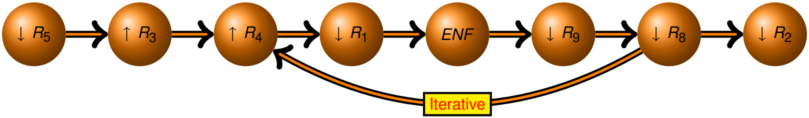


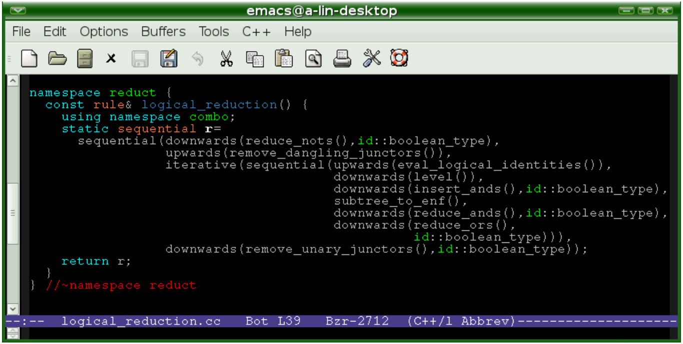


ENF: Elegant Normal Form (Holman, '90). A strategy onto itself.

- Efficient
- Retain the structure (or even improve it)

Combining the reduction rules






```
namespace reduct {
  const rule& logical_reduction() {
    using namespace combo;
    static sequential r=
      sequential(downwards(reduce_nots(),id::boolean_type),
                 upwards(remove_dangling_junctors()),
                 iterative(sequential(upwards(eval_logical_identities()),
                                      downwards(level()),
                                      downwards(insert_ands(),id::boolean_type),
                                      subtree_to_enf(),
                                      downwards(reduce_ands(),id::boolean_type),
                                      downwards(reduce_ors(),
                                                id::boolean_type))),
                 downwards(remove_unary_junctors(),id::boolean_type));
    return r;
  }
} //~namespace reduct
```

Outline

1. Introduction
   - Program Space
   - What are Programs in Normal Forms?
   - Effects of Reducing Programs in Normal Forms
2. The Reduct Toolkit
   - Recall of Combo
   - How it works
   - Demo...
   - What Remains to Be Implemented

Outline

1. Introduction
   - Program Space
   - What are Programs in Normal Forms?
   - Effects of Reducing Programs in Normal Forms
2. The Reduct Toolkit
   - Recall of Combo
   - How it works
   - Demo...
   - What Remains to Be Implemented

What remains to be implemented

- Boolean, continuous, action-perception: virtually complete

What remains to be implemented

- Boolean, continuous, action-perception: virtually complete
- Numerico-Boolean: dealing with non-linear systems like for instance `0<(*(+(#1 1) +(#1 1)))`

What remains to be implemented

- Boolean, continuous, action-perception: virtually complete
- Numerico-Boolean: dealing with non-linear systems like for instance `0<(*(+(#1 1) +(#1 1)))`
- Higher order, list, recursion, etc: everything remains to be done.

What remains to be implemented

- Boolean, continuous, action-perception: virtually complete
- Numerico-Boolean: dealing with non-linear systems like for instance `0<(*(+(#1 1) +(#1 1)))`
- Higher order, list, recursion, etc: everything remains to be done.
- Factorizing the rules, operator properties rather than operators themselves, for instance
  - `+(`X` 0)` $\rightarrow X$
  - `or(`X` false)` $\rightarrow X$

  2 instances of the *neutral_element* reduction rule.