Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
append([a,b],[c,d],Ls) =
append([X|Xs],Ys,[X|Zs])
```

Input: Two terms T_1 and T_2 to be unified

Output: Θ, the mgu of T_1 and T_2, or *failure*

Algorithm: Initialize the substitution Θ to be empty, the stack to contain the equation $T_1 = T_2$, and failure to *false*.

And then...

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

$\Theta = \{\}$

```
append([a,b],[c,d],Ls) =
append([X|Xs],Ys,[X|Zs])
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
append([a,b],[c,d],Ls) =
append([X|Xs],Ys,[X|Zs])
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
append([a,b],[c,d],Ls) =
append([X|Xs],Ys,[X|Zs])
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
append([a,b],[c,d],Ls) =
append([X|Xs],Ys,[X|Zs])
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

$\Theta = \{\}$

```
append([a,b],[c,d],Ls) =
append([X|Xs],Ys,[X|Zs])
```

```
[a,b] = [X|Xs]
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
[a,b] = [X|Xs]
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

$\Theta = \{\}$

```
[a,b] = [X|Xs]
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

`[a,b] = [X|Xs]`

`[c,d] = Ys`
`Ls = [X|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

$\Theta = \{\}$

```
[a,b] = [X|Xs]
```

```
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Excuse me?

We have this: `[a,b] = [X|Xs]`

and this: X is $f(X_1,...,X_n)$ and Y is $f(Y_1,...,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1...n$, on the stack

where exactly is the functor in `[a,b] = [X|Xs]` ???

Excuse me?

We have this: `[a,b] = [X|Xs]`

and this: X is $f(X_1,...,X_n)$ and Y is $f(Y_1,...,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1...n$, on the stack

where exactly is the functor in `[a,b] = [X|Xs]` ???

It's right there. You just can't see it. It's the dot functor or dot operator, or just the dot. It's how cons pairs are formally represented. (It's the equivalent of the cons in Haskell.)

Equivalent forms of lists

Cons pair syntax	Element syntax	Functor or dot syntax
[ ]	[ ]	[ ]
[a | [ ]]	[a]	.(a ,[ ])
[a | [b | [ ]]]	[a, b]	.(a, .(b, [ ]))
[a | [b | [c | [ ]]]]	[a, b, c]	.(a, .(b, .(c, [ ])))
[a | X]	[a | X]	.(a, X)
[a | [b | X]]	[a, b | X]	.(a, .(b, X))

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
[a,b] = [X|Xs]
```

```
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
[a,b] = [X|Xs]
```

```
a = X
[b] = Xs
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
a = X
[b] = Xs
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

$\Theta = \{\}$

```
a = X
[b] = Xs
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
a = X
```

```
[b] = Xs
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

```
a = X
```

```
[b] = Xs
[c,d] = Ys
Ls = [X|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {}

`a = X`

```
[b] = Xs
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a`}

`a = X`

```
[b] = Xs
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a`}

```
[b] = Xs
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a`}

```
[b] = Xs
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a`}

`[b] = Xs`

`[c,d] = Ys`
`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a`}

`[b] = Xs`

`[c,d] = Ys`
`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_\mathrm{i}$, $i = 1 \ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a, Xs = [b]`}

```
[b] = Xs
```

```
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b]`}

```
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop *X* = *Y* from the stack

case

X is a variable that does not occur in *Y*:
substitute *Y* for *X* in the stack and in Θ
add *X* = *Y* to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and *Y* are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and *Y* is $f(Y_1,\ldots,Y_n)$
for some functor *f* and *n* > 0:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b]`}

```
[c,d] = Ys
Ls = [a|Zs]
```

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b]`}

`[c,d] = Ys`

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b]`}

`[c,d] = Ys`

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d]`}

`[c,d] = Ys`

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d]`}

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d]`}

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop *X* = *Y* from the stack

case

X is a variable that does not occur in *Y*:
substitute *Y* for *X* in the stack and in Θ
add *X* = *Y* to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and *Y* are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and *Y* is $f(Y_1,\ldots,Y_n)$
for some functor *f* and *n* > 0:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d]`}

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1 \ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d]`}

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,...,X_n)$ and Y is $f(Y_1,..., Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1...n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

`Ls = [a|Zs]`

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]` }

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

while stack not empty and no failure *do*

pop $X = Y$ from the stack

case

X is a variable that does not occur in Y:
substitute Y for X in the stack and in Θ
add $X = Y$ to Θ

Y is a variable that does not occur in X:
substitute X for Y in the stack and in Θ
add Y = X to Θ

X and Y are identical constants or variables:
continue

X is $f(X_1,\ldots,X_n)$ and Y is $f(Y_1,\ldots,Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1\ldots n$, on the stack

otherwise:
failure is *true*

If failure, *then* output *failure else* output Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([X|Xs],Ys,[X|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

Now, if we make all the substitutions given by Θ in terms T_1 and T_2, we'll see that the two terms are identical...that is, they're unified by the unifier Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([a|Xs],Ys,[a|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

Now, if we make all the substitutions given by Θ in terms T_1 and T_2, we'll see that the two terms are identical...that is, they're unified by the unifier Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([a|[b]],Ys,[a|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

Now, if we make all the substitutions given by Θ in terms T_1 and T_2, we'll see that the two terms are identical...that is, they're unified by the unifier Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],Ls)`

T_2 = `append([a|[b]], [c,d],[a|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

Now, if we make all the substitutions given by Θ in terms T_1 and T_2, we'll see that the two terms are identical...that is, they're unified by the unifier Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d], [a|Zs])`

T_2 = `append([a|[b]],[c,d],[a|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]`}

Now, if we make all the substitutions given by Θ in terms T_1 and T_2, we'll see that the two terms are identical...that is, they're unified by the unifier Θ.

Unification Algorithm

T_1 = `append([a,b],[c,d],[a|Zs])`

T_2 = `append([a|[b]],[c,d],[a|Zs])`

failure = *false*

Θ = {`X = a,Xs = [b],`
`Ys = [c,d],Ls = [a|Zs]` }

Now, if we make all the substitutions given by Θ in terms T_1 and T_2, we'll see that the two terms are identical...that is, they're unified by the unifier Θ.

And that's how the unification algorithm works. Next, we'll see how it's used in the Prolog interpreter. Can you stand the excitement?

Questions?