Paradigmas Avanzados de Programación

Modelo de Programación Relacional

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

¿Qué es?

Relaciones (vs. funciones)

- Cero, una o más salidas.
- Cuáles son entradas y cuáles son salidas puede ser diferente en cada invocación.

Aplicaciones

- Bases de datos deductivas
- Analizadores sintácticos de gramáticas ambiguas.
- Enumeración de soluciones para problemas combinatorios complejos.

¿Qué es?

Relaciones (vs. funciones)

- Cero, una o más salidas.
- Cuáles son entradas y cuáles son salidas puede ser diferente en cada invocación.

Aplicaciones

- Bases de datos deductivas
- Analizadores sintácticos de gramáticas ambiguas.
- Enumeración de soluciones para problemas combinatorios complejos.

Nuevo concepto: Selección no-determinística

- Selección no-determinística de una opción entre varias alternativas.
- Implementada con búsqueda determinística.
- Denominada “No-determinismo no-sé”
  - Algunas escogencias pueden llevar a una solución mientras otras no; no sabemos cuál lleva a cual, pero podemos buscar una solución.
  - Contraste con el “No-determinismo no-importa”: O todas las escogencias tienen éxito o todas fallan, luego no importa cuál opción se escoge.
- Idea vieja (1967), corazón de Prolog.

Implementación de la búsqueda

- Las escogencias se realizan secuencialmente.
- Las alternativas se ensayan en el orden en que se definen.
- Se puede ilustrar con un diagrama de un árbol de búsqueda.

Cuidado con la eficiencia

- Cada escogencia nueva multiplica el tamaño del espacio de búsqueda por el número de alternativas, luego la búsqueda es exponencial.
- Es práctica cuando:
  - El espacio de búsqueda es pequeño.
  - Se usa como una herramienta exploratoria para adquirir mejor conocimiento de un problema, que pueda llevar a un algoritmo eficiente.
- Sino: necesita programación por restricciones
  - “Solvers” especializados.
  - Optimización basada en la estructura del problema.
  - Heurísticas de búsqueda.

Plan

El modelo de computación relacional

Declarativo + dos nuevas declaraciones

$\langle d \rangle$ ::=	
skip	Declaración vacía
$\langle d \rangle_1$ $\langle d \rangle_2$	Declaración de secuencia
local $\langle x \rangle$ **in** $\langle d \rangle$ **end**	Creación de variable
$\langle x \rangle_1 = \langle x \rangle_2$	ligadura variable-variable
$\langle x \rangle = \langle v \rangle$	Creación de valor
if $\langle x \rangle$ **then** $\langle d \rangle_1$ **else** $\langle d \rangle_2$ **end**	Condicional
case $\langle x \rangle$ **of** $\langle$patrón$\rangle$ **then** $\langle d \rangle_1$ **else** $\langle d \rangle_2$ **end**	Reconocimiento de patrones
{ $\langle x \rangle$ $\langle y \rangle_1 \cdots \langle y \rangle_n$ }	Invocación de procedimiento
choice $\langle d \rangle_1$ [] $\cdots$ [] $\langle d \rangle_n$ **end**	**Escogencia**
fail	**Falla**

El modelo de computación relacional

`choice` $\langle S \rangle_1$ `[]` $\cdots$ `[]` $\langle S \rangle_n$ `end`

- Hace que se ejecute provisionalmente una declaración.
- Si más tarde falla, entonces se toma otra declaración para ejecutar.
- Crea un punto de escogencia.

`fail`

- Indica que la alternativa actual es fallida.
- El fallo es implícito cuando se trata de ligar dos valores incompatibles (3=4).

El modelo de computación relacional

`choice` $\langle S \rangle_1$ `[]` $\cdots$ `[]` $\langle S \rangle_n$ `end`

- Hace que se ejecute provisionalmente una declaración.
- Si más tarde falla, entonces se toma otra declaración para ejecutar.
- Crea un punto de escogencia.

`fail`

- Indica que la alternativa actual es fallida.
- El fallo es implícito cuando se trata de ligar dos valores incompatibles (3=4).

Ejemplo: Combinación de colores

```
declare
fun {Claro} choice blanco [] habano end end
fun {Oscuro} choice negro [] azul end end

proc {Contraste P1 P2}
   choice P1={Claro} P2={Oscuro} [] P1={Oscuro} P2={Claro} end
end

fun {Pinta}
   Camisa Pantalón Medias
in
   {Contraste Camisa Pantalón}
   {Contraste Pantalon Medias}
   if Camisa==Medias then fail end
   pinta(Camisa Pantalón Medias)
end
```

El modelo de computación relacional

Implementación del `choice`

- Encapsular la información necesitada para volver atrás hasta el útlimo punto de escogencia y escoger una alternativa diferente.
- O, continuar la ejecución en un espacio de computación hijo que copia información del espacio padre, y lo ejecuta aisladamente.
- Oz utiliza espacios de computación

Árboles de búsqueda

- Cada nodo corresponde a un punto de escogencia.
- Cada subárbol corresponde a una alternativa.
- Cada camino de la raíz a una hoja corresponde a una posible secuencia de ejecución.
- Un camino puede terminar en éxito o en falla.
- Un árbol de búsqueda muestra todos los caminos al tiempo (fallidos y exitosos).

Búsqueda encapsulada

- La ejecución se lleva a cabo en un ambiente aislado, de manera que el resto del programa no se afecta por lo que pase en el espacio encapsulado.
- Modularidad: Puede haber más de un programa relacional ejecutándose concurrentemente.
- Composicionalidad: una búsqueda encapsulada puede estar anidada dentro de otra búsqueda encapsulada.
- Facilita ejecutar el mismo programa en diferentes formas:
  - Especificación de la estrategia de búsqueda: Profundidad, amplitud, ...
  - Especificación del número o tipo de solución deseada: una, todas, las mejores, ...

El modelo de computación relacional

Nuevo concepto: Espacios de Computación

- Un hilo y un almacén que puede contener otros hilos.
- Se usa para implementar **`choice`**, **`fail`** y `Solve`.

Miremos primero otros ejemplos . . .

- Ejemplos numéricos
- El producto palíndrome

El modelo de computación relacional

Nuevo concepto: Espacios de Computación

- Un hilo y un almacén que puede contener otros hilos.
- Se usa para implementar **`choice`**, **`fail`** y `Solve`.

Miremos primero otros ejemplos . . .

- Ejemplos numéricos
- El producto palíndrome

Plan

El modelo de computación relacional

Ejemplo: Números de un dígito

```
fun {Dígito}
   choice 0 [] 1 [] 2 [] 3 [] 4 [] 5 [] 6 [] 7 [] 8 [] 9 end
end
```

Ejemplo: Números de dos dígitos

```
fun {DosDígitos}
   10*{Dígito}+{Dígito}
end
```

Ejemplo: Números de dos dígitos, forma extraña

```
fun {DosDígitosExtraña}
   {Dígito}+ 10*{Dígito}
end
```

El modelo de computación relacional

Ejemplo: Números de un dígito

```
fun {Dígito}
   choice 0 [] 1 [] 2 [] 3 [] 4 [] 5 [] 6 [] 7 [] 8 [] 9 end
end
```

Ejemplo: Números de dos dígitos

```
fun {DosDígitos}
   10*{Dígito}+{Dígito}
end
```

Ejemplo: Números de dos dígitos, forma extraña

```
fun {DosDígitosExtraña}
   {Dígito}+ 10*{Dígito}
end
```

Ejemplo: Números de un dígito

```
fun {Dígito}
   choice 0 [] 1 [] 2 [] 3 [] 4 [] 5 [] 6 [] 7 [] 8 [] 9 end
end
```

Ejemplo: Números de dos dígitos

```
fun {DosDígitos}
   10*{Dígito}+{Dígito}
end
```

Ejemplo: Números de dos dígitos, forma extraña

```
fun {DosDígitosExtraña}
   {Dígito}+ 10*{Dígito}
end
```

El modelo de computación relacional

Ejemplo: Producto palíndrome

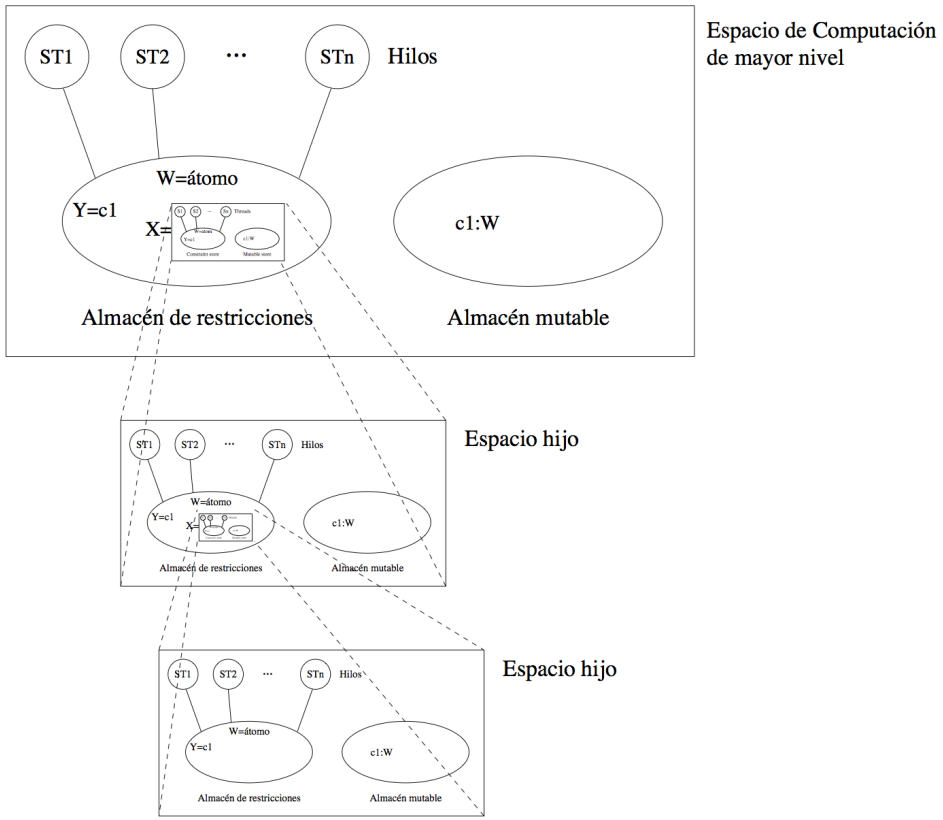
Encontrar los números palíndromes de cuatro dígitos, que sean el producto de dos números de dos dígitos.

```
proc {Palíndrome ?X}
   X=(10*{Dígito}+{Dígito}) * (10*{Dígito}+{Dígito})
   (X>0)=true
   (X>1000)=true
   (X div 1000) mod 10 = (X div 1) mod 10
   (X div 100) mod 10 = (X div 10) mod 10
end
```

Plan

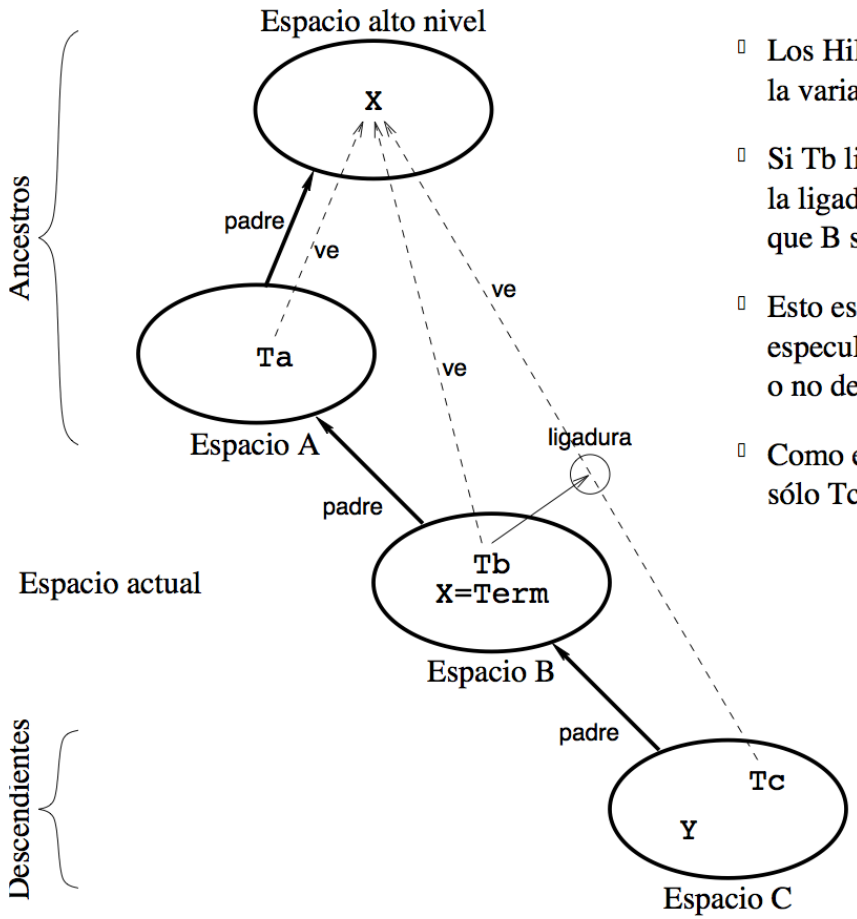
El modelo de computación relacional

Espacios de Computación: modelo



El modelo de computación relacional

Espacios de Computación: visibilidad



- Los Hilos Ta, Tb, y Tc ven la variable X.
- Si Tb liga X entonces Tb & Tc verán la ligadura. Ta no la verá a menos que B sea mezclado en A.
- Esto es porque los espacios hijos son especulativos: ellos pueden hacer parte o no del almacén del padre.
- Como el espacio C es especulativo, sólo Tc ve Y (Ta y Tb no la ven).

Espacios de Computación: Operaciones

$$
\begin{array}{rl}
\langle \text{declaración} \rangle ::= & \{\texttt{NewSpace}\ \langle x \rangle\ \langle y \rangle\} \\
\mid & \{\texttt{WaitStable}\} \\
\mid & \{\texttt{Choose}\ \langle x \rangle\ \langle y \rangle\} \\
\mid & \{\texttt{Ask}\ \langle x \rangle\ \langle y \rangle\} \\
\mid & \{\texttt{Commit}\ \langle x \rangle\ \langle y \rangle\} \\
\mid & \{\texttt{Clone}\ \langle x \rangle\ \langle y \rangle\} \\
\mid & \{\texttt{Inject}\ \langle x \rangle\ \langle y \rangle\} \\
\mid & \{\texttt{Merge}\ \langle x \rangle\ \langle y \rangle\}
\end{array}
$$

El modelo de computación relacional

Espacios de Computación: `NewSpace`

`S={NewSpace P}`, recibe un procedimiento `P` de un argumento, crea un espacio de computación nuevo y devuelve una referencia a él. En este espacio se crean una variable fresca `R` y un hilo nuevo, y se invoca `{P R}` en el hilo.

Espacios de Computación: `WaitStable`

Espera hasta que el espacio actual se vuelve estable. EN ese momento, el hilo que contiene `WaitStable` se vuelve ejecutable de nuevo. Este hilo ejecutará finalmente una operación `Choose`. Visto desde afuera, el espacio no es estable hasta que la operación se encuentre la operación `Choose`.

El modelo de computación relacional

Espacios de Computación: `NewSpace`

`S={NewSpace P}`, recibe un procedimiento `P` de un argumento, crea un espacio de computación nuevo y devuelve una referencia a él. En este espacio se crean una variable fresca `R` y un hilo nuevo, y se invoca `{P R}` en el hilo.

Espacios de Computación: `WaitStable`

Espera hasta que el espacio actual se vuelve estable. EN ese momento, el hilo que contiene `WaitStable` se vuelve ejecutable de nuevo. Este hilo ejecutará finalmente una operación `Choose`. Visto desde afuera, el espacio no es estable hasta que la operación se encuentre la operación `Choose`.

Espacios de Computación: `Choose`

- `Y={Choose N}` espera hasta que el espacio actual esté estable (normalmente debería estarlo), bloquea el hilo actual, y crea un punto de escogencia con `N` alternativas en el espacio actual.
- El `Choose` espera que se escoja una alternativa por medio de una operación `Commit` sobre el espacio.
- La invocación a `Choose` sólo define *cuántas* alternativas hay: no especifica qué hacer en cada una de ellas.
- Finalmente, el `Choose` continúa su ejecución, ligando `Y=I` cuando se haya escogido la alternativa `I` ($1 \leq I \leq N$).

Espacios de Computación: Estados

- Un espacio de computación es ejecutable si él o un descendiente contiene algún hilo ejecutable. Sino, el espacio está bloqueado.
- Una vez se ejecuten todos los hilos del espacio y sus descendientes hasta que quede bloqueado, el espacio quedará en uno de los estados siguientes:
  - Estable: ninguna información adicional añadida por sus ancestros lo volverá ejecutable.
  - Suspendido: alguna información adicional añadida por un ancestro lo puede volver ejecutable. Normalmente, estar suspendido es una condición temporal debido a la concurrencia.

Espacios de Computación: Estados estables

Un estado estable puede estar en cuatro estados:

- Éxito (succeeded): No hay puntos de escogencia. Contiene una solución a un programa lógico.
- Distribuible (distributable): El espacio tiene un hilo suspendido en un punto de escogencia con dos o más alternativas.
- Fallido (falied): Se produjo alguna inconsistencia (e.g. ligar una variable con dos valores diferentes).
- Mezclado (merged): El espacio ha sido descartado y su almacén fue mezclado con el del espacio padre.

El modelo de computación relacional

Espacios de Computación: `Ask`

- `A={Ask S}` averigua el estado del espacio `S`.
- `A` se liga tan pronto el estado se vuelve estable.
- Si `S` es failed, merged, o succeeded, entonces `Ask` devuelve `failed`, `merged`, o `succeeded`.
- Si `S` es distribuible, entonces devuelve `alternatives(N)`, donde `N` es el número de alternativas.

El modelo de computación relacional

Espacios de Computación: `Clone`

`C={Clone S}` espera hasta que `S` sea estable, y entonces crea una copia idéntica de `S` y devuelve una referencia a la copia. Esto es lo que permite que se puedan explorar varias alternativas.

Espacios de Computación: `Commit`

Si `S` es un espacio distribuible, `{Commit S I}` hace que se complete la invocación de `Choose` en el espacio y devuelva `I` como resultado. Esto puede hacer que el espacio reanude su ejecución.

El modelo de computación relacional

Espacios de Computación: `Clone`

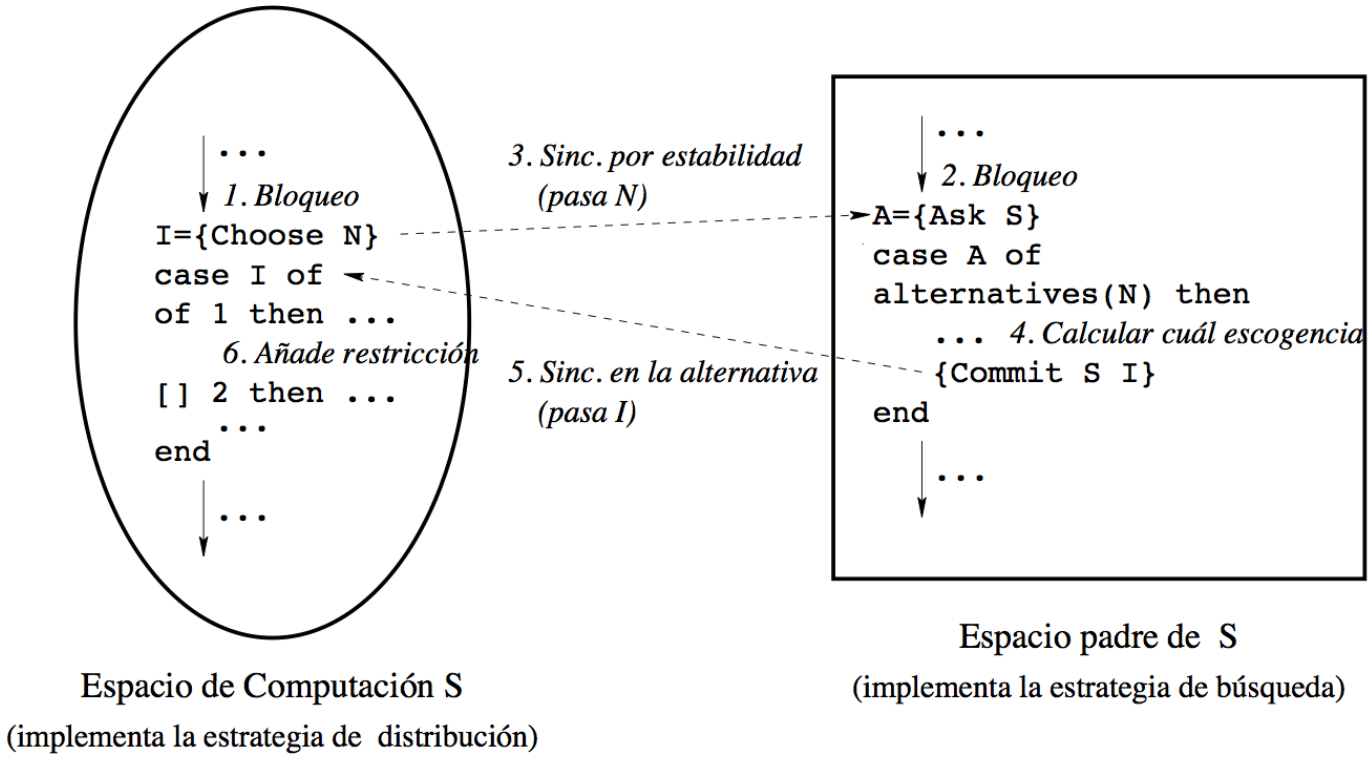
`C={Clone S}` espera hasta que `S` sea estable, y entonces crea una copia idéntica de `S` y devuelve una referencia a la copia. Esto es lo que permite que se puedan explorar varias alternativas.

Espacios de Computación: `Commit`

Si `S` es un espacio distribuible, `{Commit S I}` hace que se complete la invocación de `Choose` en el espacio y devuelva `I` como resultado. Esto puede hacer que el espacio reanude su ejecución.

El modelo de computación relacional

Espacios de Computación: comunicación estrategias de exploración y de búsqueda



El modelo de computación relacional

Espacios de Computación: `Merge`

`{Merge S Y}` liga `Y` con la variable raíz del espacio y descarta el espacio.

Espacios de Computación: `Inject`

`{Inject S P}` es similar a la creación de un espacio, sólo que usa un espacio existente. Crea un nuevo hilo en ese espacio e invoca `{P R}`, donde `R` es la variable raíz del espacio.

El modelo de computación relacional

Espacios de Computación: `Merge`

`{Merge S Y}` liga `Y` con la variable raíz del espacio y descarta el espacio.

Espacios de Computación: `Inject`

`{Inject S P}` es similar a la creación de un espacio, sólo que usa un espacio existente. Crea un nuevo hilo en ese espacio e invoca `{P R}`, donde `R` es la variable raíz del espacio.

Implementación del modelo de computación relacional

¿Qué es el modelo relacional?

El modelo declarativo extendido con las declaraciones **`choice`** y **`fail`**, y la operación `Solve` para hacer búsqueda encapsulada.

Implementación de `choice`

`choice` ⟨stmt⟩$_1$ `[]` ⟨stmt⟩$_2$ `[]` `...` `[]` ⟨stmt⟩$_n$ **`end`**

es una abstracción lingüística de

```
case {Choose N}
of 1 then ⟨stmt⟩1
[] 2 then ⟨stmt⟩2
...
[] N then ⟨stmt⟩n
end
```

Implementación del modelo de computación relacional

¿Qué es el modelo relacional?

El modelo declarativo extendido con las declaraciones **`choice`** y **`fail`**, y la operación `Solve` para hacer búsqueda encapsulada.

Implementación de `choice`

`choice` $\langle \text{stmt} \rangle_1$ `[]` $\langle \text{stmt} \rangle_2$ `[]` `...` `[]` $\langle \text{stmt} \rangle_n$ **`end`**

es una abstracción lingüística de

```
case {Choose N}
of 1 then ⟨stmt⟩1
[] 2 then ⟨stmt⟩2
...
[] N then ⟨stmt⟩n
end
```

Implementación del modelo de computación relacional

Búsqueda de soluciones: `Solve` (1)

```
fun {Solve Script}
   {SolveStep {Space.new Script} nil}
end

fun {SolveStep S SolTail}
   case {Space.ask S}
   of failed           then SolTail
   [] succeeded        then {Space.merge S}|SolTail
   [] alternatives(N)  then {SolveLoop S 1 N SolTail}
   end
end
```

Búsqueda de soluciones: `Solve` (2)

```
fun lazy {SolveLoop S I N SolTail}
   if I>N then
      SolTail
   elseif I==N then
      {Space.commit S I}
      {SolveStep S SolTail}
   else
      C={Space.clone S}
      NewTail={SolveLoop S I+1 N SolTail}
   in
      {Space.commit C I}
      {SolveStep C NewTail}
   end
end
```

Modelos y Paradigmas de Programación

Universidad del Valle

Implementación del modelo de computación relacional

Ejemplo: `{Solve Pinta}`