Paradigmas Fundamentales de Programación

Concurrencia dirigida por la demanda

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

Plan

Plan

Ejecución perezosa

¿Existen otras estrategias de ejecución para los programas declarativos?

- Estándar: Evaluación ansiosa o dirigida por los datos.
- Evaluación perezosa: una declaración sólo se ejecuta cuando su resultado se necesita en alguna otra parte del programa.
- `F1`, `F2`, y `F3` son perezosas.
- Se crean "ejecuciones detenidas" que continuarán sólo cuando se necesiten sus resultados.
- Solo se necesitan `A` y `B`.

Por ejemplo:

```
fun lazy {F1 X}
   1+X*(3+X*(3+X))
end
fun lazy {F2 X}
   Y=X*X in Y*Y
end
fun lazy {F3 X}
   (X+1)*(X+1)
end
A={F1 10}
B={F2 20}
C={F3 30}
D=A+B
```

Ejecución perezosa

¿Existen otras estrategias de ejecución para los programas declarativos?

- Estándar: Evaluación ansiosa o dirigida por los datos.
- Evaluación perezosa: una declaración sólo se ejecuta cuando su resultado se necesita en alguna otra parte del programa.
- `F1`, `F2`, y `F3` son perezosas.
- Se crean "ejecuciones detenidas" que continuarán sólo cuando se necesiten sus resultados.
- Solo se necesitan `A` y `B`.

Por ejemplo:

```
fun lazy {F1 X}
   1+X*(3+X*(3+X))
end
fun lazy {F2 X}
   Y=X*X in Y*Y
end
fun lazy {F3 X}
   (X+1)*(X+1)
end
A={F1 10}
B={F2 20}
C={F3 30}
D=A+B
```

Ejecución perezosa

¿Existen otras estrategias de ejecución para los programas declarativos?

- Estándar: Evaluación ansiosa o dirigida por los datos.
- Evaluación perezosa: una declaración sólo se ejecuta cuando su resultado se necesita en alguna otra parte del programa.
- `F1`, `F2`, y `F3` son perezosas.
- Se crean "ejecuciones detenidas" que continuarán sólo cuando se necesiten sus resultados.
- Solo se necesitan `A` y `B`.

Por ejemplo:

```
fun lazy {F1 X}
   1+X*(3+X*(3+X))
end
fun lazy {F2 X}
   Y=X*X in Y*Y
end
fun lazy {F3 X}
   (X+1)*(X+1)
end
A={F1 10}
B={F2 20}
C={F3 30}
D=A+B
```

Ejecución perezosa

¿Existen otras estrategias de ejecución para los programas declarativos?

- Estándar: Evaluación ansiosa o dirigida por los datos.
- Evaluación perezosa: una declaración sólo se ejecuta cuando su resultado se necesita en alguna otra parte del programa.
- `F1`, `F2`, y `F3` son perezosas.
- Se crean "ejecuciones detenidas" que continuarán sólo cuando se necesiten sus resultados.
- Solo se necesitan `A` y `B`.

Por ejemplo:

```
fun lazy {F1 X}
   1+X*(3+X*(3+X))
end
fun lazy {F2 X}
   Y=X*X in Y*Y
end
fun lazy {F3 X}
   (X+1)*(X+1)
end
A={F1 10}
B={F2 20}
C={F3 30}
D=A+B
```

Ejecución perezosa

¿Existen otras estrategias de ejecución para los programas declarativos?

- Estándar: Evaluación ansiosa o dirigida por los datos.
- Evaluación perezosa: una declaración sólo se ejecuta cuando su resultado se necesita en alguna otra parte del programa.
- `F1`, `F2`, y `F3` son perezosas.
- Se crean "ejecuciones detenidas" que continuarán sólo cuando se necesiten sus resultados.
- Solo se necesitan `A` y `B`.

Por ejemplo:

```
fun lazy {F1 X}
   1+X*(3+X*(3+X))
end
fun lazy {F2 X}
   Y=X*X in Y*Y
end
fun lazy {F3 X}
   (X+1)*(X+1)
end
A={F1 10}
B={F2 20}
C={F3 30}
D=A+B
```

Ejecución perezosa

¿Existen otras estrategias de ejecución para los programas declarativos?

- Estándar: Evaluación ansiosa o dirigida por los datos.
- Evaluación perezosa: una declaración sólo se ejecuta cuando su resultado se necesita en alguna otra parte del programa.
- `F1`, `F2`, y `F3` son perezosas.
- Se crean "ejecuciones detenidas" que continuarán sólo cuando se necesiten sus resultados.
- Solo se necesitan `A` y `B`.

Por ejemplo:

```
fun lazy {F1 X}
   1+X*(3+X*(3+X))
end
fun lazy {F2 X}
   Y=X*X in Y*Y
end
fun lazy {F3 X}
   (X+1)*(X+1)
end
A={F1 10}
B={F2 20}
C={F3 30}
D=A+B
```

Plan

El modelo concurrente dirigido por la demanda

El modelo concurrente dirigido por la demanda extiende el modelo concurrente dirigido por los datos con un solo concepto nuevo, el disparador by-need.

⟨d⟩ ::=	
skip	Declaración vacía
⟨d⟩$_1$ ⟨d⟩$_2$	Declaración de secuencia
local ⟨x⟩ **in** ⟨d⟩ **end**	Creación de variable
⟨x⟩$_1$=⟨x⟩$_2$	Ligadura variable-variable
⟨x⟩=⟨v⟩	Creación de valor
if ⟨x⟩ **then** ⟨d⟩$_1$ **else** ⟨d⟩$_2$ **end**	Condicional
case ⟨x⟩ **of** ⟨patrón⟩ **then** ⟨d⟩$_1$ **else** ⟨d⟩$_2$ **end**	Reconocimiento de patrones
{ ⟨x⟩ ⟨y⟩$_1$ ··· ⟨y⟩$_n$ }	Invocación de procedimiento
thread ⟨d⟩ **end**	Creación de hilo
{ByNeed ⟨x⟩ ⟨y⟩ }	**Creación de disparador**

Plan

El disparador by-need

Intuición de `ByNeed`

- La declaración `{ByNeed P Y}` tiene el mismo efecto que la declaración **`thread`** `{P Y}` **`end`**, salvo para la planificación.
- Ambas declaraciones invocan el procedimiento `P` en su propio hilo con argumento `Y`.
- La diferencia entre las declaraciones es el momento en que la invocación al procedimiento se ejecuta.
- **`thread`** `{P Y}` **`end`**: `{P Y}` se ejecutará finalmente, siempre.
- `{ByNeed P Y}`: `{P Y}` se ejecutará sólo si se necesita el valor de `Y`.

Ejemplo

```
{ByNeed proc {$ A}
          A=111*111
        end
 Y}
{Browse Y}
```

El disparador by-need

Intuición de `ByNeed`

- La declaración `{ByNeed P Y}` tiene el mismo efecto que la declaración **`thread`** `{P Y}` **`end`**, salvo para la planificación.
- Ambas declaraciones invocan el procedimiento `P` en su propio hilo con argumento `Y`.
- La diferencia entre las declaraciones es el momento en que la invocación al procedimiento se ejecuta.
- **`thread`** `{P Y}` **`end`**: `{P Y}` se ejecutará finalmente, siempre.
- `{ByNeed P Y}`: `{P Y}` se ejecutará sólo si se necesita el valor de `Y`.

Ejemplo

```
{ByNeed proc {$ A}
          A=111*111
        end
        Y}
{Browse Y}
```

El disparador by-need

Intuición de `ByNeed`

- La declaración `{ByNeed P Y}` tiene el mismo efecto que la declaración **`thread`** `{P Y}` **`end`**, salvo para la planificación.
- Ambas declaraciones invocan el procedimiento `P` en su propio hilo con argumento `Y`.
- La diferencia entre las declaraciones es el momento en que la invocación al procedimiento se ejecuta.
- **`thread`** `{P Y}` **`end`**: `{P Y}` se ejecutará finalmente, siempre.
- `{ByNeed P Y}`: `{P Y}` se ejecutará sólo si se necesita el valor de `Y`.

Ejemplo

```
{ByNeed proc {$ A}
          A=111*111
        end
 Y}
{Browse Y}
```

El disparador by-need

Intuición de `ByNeed`

- La declaración `{ByNeed P Y}` tiene el mismo efecto que la declaración **`thread`** `{P Y}` **`end`**, salvo para la planificación.
- Ambas declaraciones invocan el procedimiento `P` en su propio hilo con argumento `Y`.
- La diferencia entre las declaraciones es el momento en que la invocación al procedimiento se ejecuta.
- **`thread`** `{P Y}` **`end`**: `{P Y}` se ejecutará finalmente, siempre.
- `{ByNeed P Y}`: `{P Y}` se ejecutará sólo si se necesita el valor de `Y`.

Ejemplo

```
{ByNeed proc {$ A}
          A=111*111
        end
        Y}
{Browse Y}
```

El disparador by-need

Intuición de `ByNeed`

- La declaración `{ByNeed P Y}` tiene el mismo efecto que la declaración **`thread`** `{P Y}` **`end`**, salvo para la planificación.
- Ambas declaraciones invocan el procedimiento `P` en su propio hilo con argumento `Y`.
- La diferencia entre las declaraciones es el momento en que la invocación al procedimiento se ejecuta.
- **`thread`** `{P Y}` **`end`**: `{P Y}` se ejecutará finalmente, siempre.
- `{ByNeed P Y}`: `{P Y}` se ejecutará sólo si se necesita el valor de `Y`.

Ejemplo

```
{ByNeed proc {$ A}
          A=111*111
        end
        Y}
{Browse Y}
```

El disparador by-need

Intuición de `ByNeed`

- La declaración `{ByNeed P Y}` tiene el mismo efecto que la declaración **`thread`** `{P Y}` **`end`**, salvo para la planificación.
- Ambas declaraciones invocan el procedimiento `P` en su propio hilo con argumento `Y`.
- La diferencia entre las declaraciones es el momento en que la invocación al procedimiento se ejecuta.
- **`thread`** `{P Y}` **`end`**: `{P Y}` se ejecutará finalmente, siempre.
- `{ByNeed P Y}`: `{P Y}` se ejecutará sólo si se necesita el valor de `Y`.

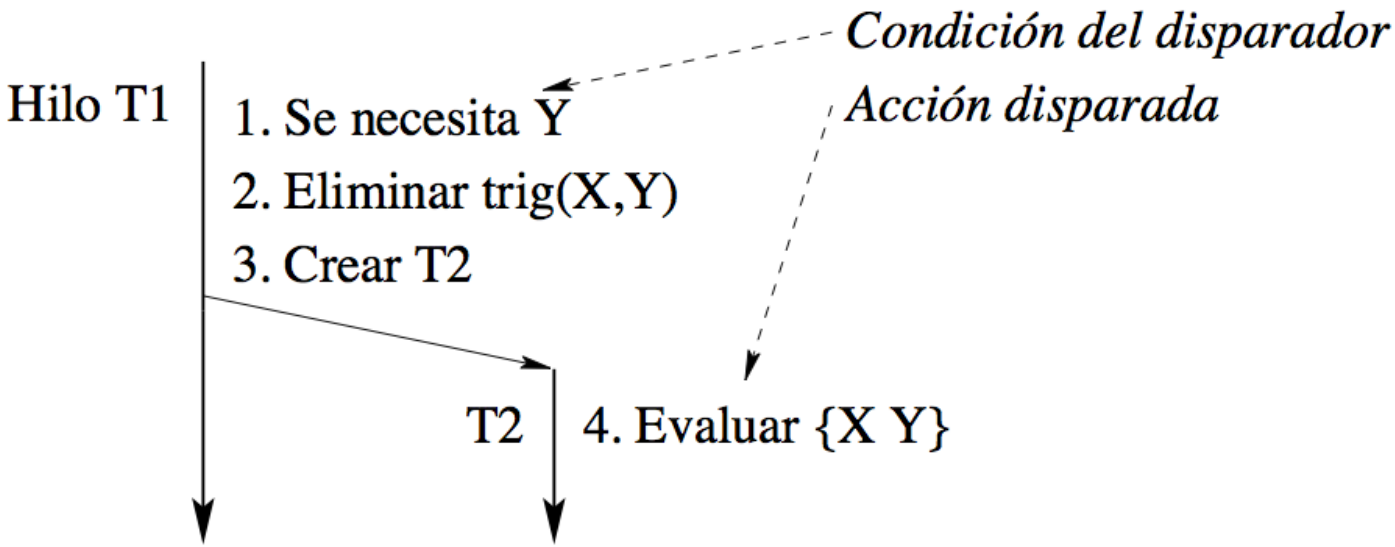
Ejemplo

```
{ByNeed proc {$ A}
          A=111*111
        end
        Y}
{Browse Y}
```

Plan

Semántica de los disparadores by-need (1)

El protocolo

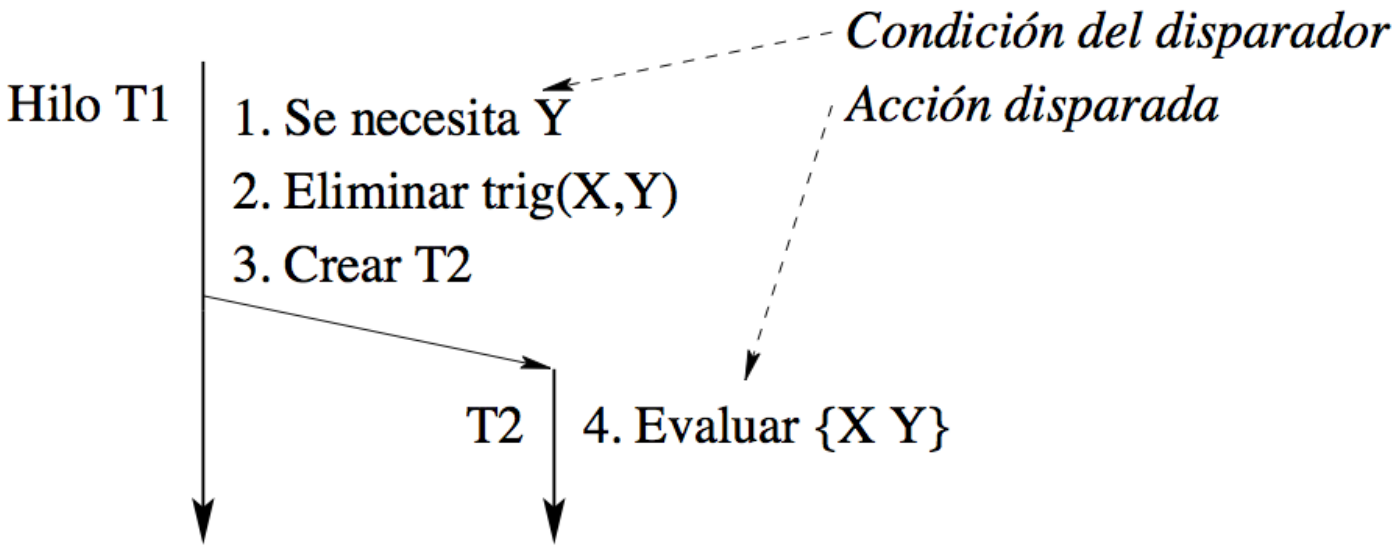


Disparador

- (Condición de activación, acción)
- Activación del disparador: Cuando se cumple la condición, se ejecuta la acción por una vez.
- Disparador by-need: la condición de activación es la necesidad de una variable.

Semántica de los disparadores by-need (1)

El protocolo

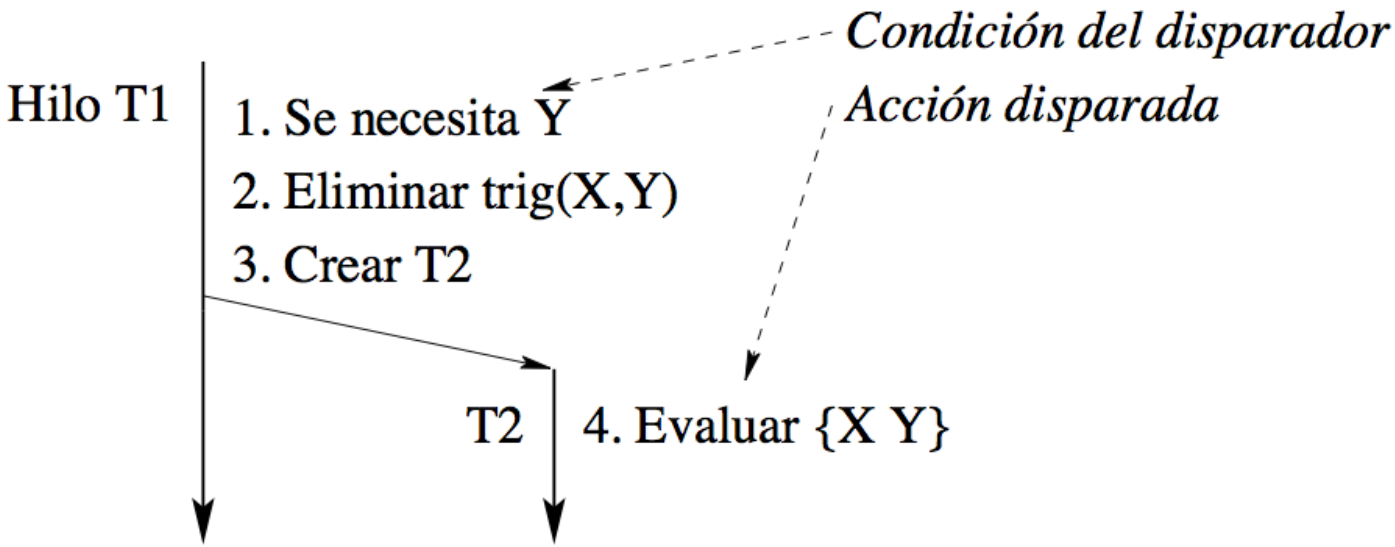


Disparador

- (Condición de activación, acción)
- Activación del disparador: Cuando se cumple la condición, se ejecuta la acción por una vez.
- Disparador by-need: la condición de activación es la necesidad de una variable.

Semántica de los disparadores by-need (1)

El protocolo

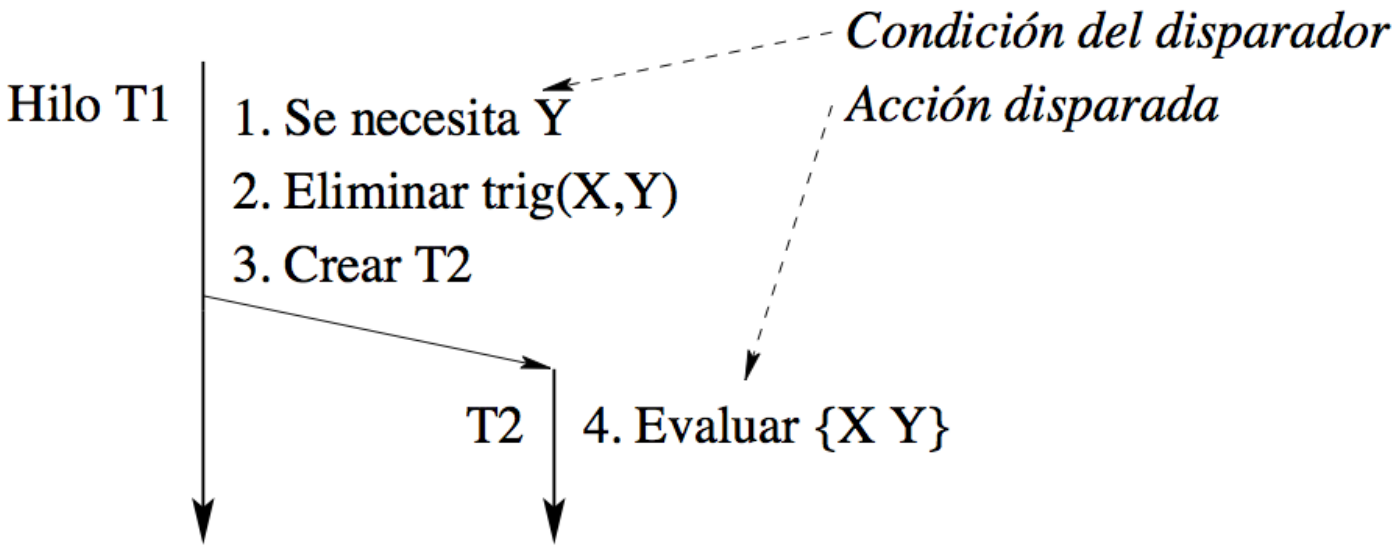


Disparador

- (Condición de activación, acción)
- Activación del disparador: Cuando se cumple la condición, se ejecuta la acción por una vez.
- Disparador by-need: la condición de activación es la necesidad de una variable.

Semántica de los disparadores by-need (1)

El protocolo



Disparador

- (Condición de activación, acción)
- Activación del disparador: Cuando se cumple la condición, se ejecuta la acción por una vez.
- Disparador by-need: la condición de activación es la necesidad de una variable.

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

Creación de disparadores by-need

La declaración semántica es

$(\{\texttt{ByNeed}\ \langle x\rangle\ \langle y\rangle\},E)$

La ejecución consiste de:

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo σ y almacén de disparadores τ.

Creación de disparadores by-need

La declaración semántica es

$(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$

La ejecución consiste de:

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo σ_T: almacén de disparadores.
- Estado de la ejecución: (MST, σ_T, σ), inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es

$(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$

La ejecución consiste de:

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo τ: almacén de disparadores.
- Estado de la ejecución: (MST, σ, τ). τ inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es $(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$

La ejecución consiste de:

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo τ: almacén de disparadores.
- Estado de la ejecución: (MST, σ, τ). τ inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es

$(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$

La ejecución consiste de:

- Si $E(\langle y \rangle)$ no está determinada, entonces agregue el disparador $trig(E(\langle x \rangle), E(\langle y \rangle))$ al almacén de disparadores.

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo τ: almacén de disparadores.
- Estado de la ejecución: (MST, σ, τ). τ inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es $(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$

La ejecución consiste de:

- Si $E(y)$ no está determinada, entonces agregue el disparador $trig(E(x), E(y))$ al almacén de disparadores.
- Sino, si $E(y)$ está determinada, entonces cree un hilo nuevo con la declaración semántica inicial $(\{\langle x \rangle\ \langle y \rangle\}, E)$

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo τ: almacén de disparadores.
- Estado de la ejecución: (MST, σ, τ). τ inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es
$(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$
La ejecución consiste de:

- Si $E(\langle y \rangle)$ no está determinada, entonces agregue el disparador $trig(E(\langle x \rangle), E(\langle y \rangle))$ al almacén de disparadores.
- Sino, si $E(\langle y \rangle)$ está determinada, entonces cree un hilo nuevo con la declaración semántica inicial $(\{\langle x \rangle\ \langle y \rangle\}, E)$.

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo τ: almacén de disparadores.
- Estado de la ejecución: (MST, σ, τ). τ inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es
$(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$
La ejecución consiste de:

- Si $E(\langle y \rangle)$ no está determinada, entonces agregue el disparador $trig(E(\langle x \rangle), E(\langle y \rangle))$ al almacén de disparadores.
- Sino, si $E(\langle y \rangle)$ está determinada, entonces cree un hilo nuevo con la declaración semántica inicial $(\{\langle x \rangle\ \langle y \rangle\}, E)$.

Semántica de los disparadores by-need (2)

Semántica en tres etapas

- Agregamos un almacén de disparadores al estado de la ejecución.
- Definimos dos operaciones, creación y activación de un disparador.
- Precisamos lo que significa "necesitar" una variable.

Extensión del estado de la ejecución

- Disparador by-need: $trig(x, y)$ donde y es una variable de flujo de datos, y x es un procedimiento de un argumento.
- Un almacén nuevo τ: almacén de disparadores.
- Estado de la ejecución: (MST, σ, τ). τ inicialmente vacío.

Creación de disparadores by-need

La declaración semántica es
$(\{\texttt{ByNeed}\ \langle x \rangle\ \langle y \rangle\}, E)$
La ejecución consiste de:

- Si $E(\langle y \rangle)$ no está determinada, entonces agregue el disparador $trig(E(\langle x \rangle), E(\langle y \rangle))$ al almacén de disparadores.
- Sino, si $E(\langle y \rangle)$ está determinada, entonces cree un hilo nuevo con la declaración semántica inicial $(\{\langle x \rangle\ \langle y \rangle\}, E)$.

Semántica de los disparadores by-need (3)

Activación de disparadores by-need

Si $trig(x, y) \in \tau$ y se necesita y (hilo suspendido esperando que y se determine, o intento de ligar y para determinarla):

- Elimine el disparador de τ.
- Cree un hilo nuevo:
  $(\{\langle x\rangle\ \langle y\rangle\}, \{\langle x\rangle \rightarrow x, \langle y\rangle \rightarrow y\})$

Necesidad de una variable

- La definición debe conservar declaratividad.
- Una variable es necesitada por una operación suspendida si ésta debe estar determinada para que la operación continúe.

```
thread X={ByNeed
          fun {$} 3 end}
end
```

```
thread Y={ByNeed
          fun {$} 4 end}
end
thread Z=X+Y end
```

¿Todas las ejecuciones posibles son equivalentes?

Necesidad de una variable (cont).

Semántica de los disparadores by-need (3)

Activación de disparadores by-need

Si $trig(x, y) \in \tau$ y se necesita y (hilo suspendido esperando que y se determine, o intento de ligar y para determinarla):

- Elimine el disparador de τ.
- Cree un hilo nuevo:
  $(\{\langle x\rangle\ \langle y\rangle\}, \{\langle x\rangle \rightarrow x, \langle y\rangle \rightarrow y\})$

Necesidad de una variable

- La definición debe conservar declaratividad.
- Una variable es necesitada por una operación suspendida si ésta debe estar determinada para que la operación continúe.

```
thread X={ByNeed
          fun {$} 3 end}
end
```

```
thread Y={ByNeed
          fun {$} 4 end}
end
thread Z=X+Y end
```

¿Todas las ejecuciones posibles son equivalentes?

Necesidad de una variable (cont).

- Una variable determinada se necesita.

```
thread X={ByNeed
          fun {$} 3 end}
end
thread X=2 end
thread Z=X+4 end
```

¿Fallan todas las ejecuciones posibles?

Semántica de los disparadores by-need (3)

Activación de disparadores by-need

Si $trig(x, y) \in \tau$ y se necesita y (hilo suspendido esperando que y se determine, o intento de ligar y para determinarla):

- Elimine el disparador de τ.
- Cree un hilo nuevo:
  $(\{\langle x\rangle\ \langle y\rangle\}, \{\langle x\rangle \rightarrow x, \langle y\rangle \rightarrow y\})$

Necesidad de una variable

- La definición debe conservar declaratividad.
- Una variable es necesitada por una operación suspendida si ésta debe estar determinada para que la operación continúe.

```
thread X={ByNeed
          fun {$} 3 end}
end
```

```
thread Y={ByNeed
          fun {$} 4 end}
end
thread Z=X+Y end
```

¿Todas las ejecuciones posibles son equivalentes?

Necesidad de una variable (cont).

- Una variable determinada se necesita.

```
thread X={ByNeed
          fun {$} 3 end}
end
thread X=2 end
thread Z=X+4 end
```

¿Fallan todas las ejecuciones posibles?

Semántica de los disparadores by-need (3)

Activación de disparadores by-need

Si $trig(x, y) \in \tau$ y se necesita y (hilo suspendido esperando que y se determine, o intento de ligar y para determinarla):

- Elimine el disparador de τ.
- Cree un hilo nuevo:
  $(\{\langle x\rangle\ \langle y\rangle\}, \{\langle x\rangle \rightarrow x, \langle y\rangle \rightarrow y\})$

Necesidad de una variable

- La definición debe conservar declaratividad.
- Una variable es necesitada por una operación suspendida si ésta debe estar determinada para que la operación continúe.

```
thread X={ByNeed
          fun {$} 3 end}
end
```

```
thread Y={ByNeed
          fun {$} 4 end}
end
thread Z=X+Y end
```

¿Todas las ejecuciones posibles son equivalentes?

Necesidad de una variable (cont).

- Una variable determinada se necesita.

```
thread X={ByNeed
          fun {$} 3 end}
end
thread X=2 end
thread Z=X+4 end
```

¿Fallan todas las ejecuciones posibles?

Plan

Implementación de funciones perezosas

- Recuerde

```
fun lazy {Generate N} N|{Generate N+1} end
 L={Generate 0}
 {Browse L}
```

Esto no desplegará nada hasta que `L` sea necesitada: Browse L.2.2.1

- ¿Cómo se implementa esta abstracción lingüística?

```
fun {Generate N}
   {ByNeed fun {$} N|{Generate N+1} end}
end
```

El cuerpo ha sido empaquetado en una función de cero argumentos que sólo es invocada cuando el valor de `{Generate N}` se necesite.

Implementación de funciones perezosas

- Recuerde

```
fun lazy {Generate N} N|{Generate N+1} end
 L={Generate 0}
 {Browse L}
```

Esto no desplegará nada hasta que `L` sea necesitada: Browse L.2.2.1

- ¿Cómo se implementa esta abstracción lingüística?

```
fun {Generate N}
   {ByNeed fun {$} N|{Generate N+1} end}
end
```

El cuerpo ha sido empaquetado en una función de cero argumentos que sólo es invocada cuando el valor de `{Generate N}` se necesite.

Plan

Modelos de computación declarativos y prácticos

	secuencial con valores	*secuencial con valores y vars. de flujo de datos*	*concurrente con valores y vars. de flujo de datos*
ejecución ansiosa (estricta)	programación funcional estricta (e.g., Scheme, ML) (1)&(2)&(3)	modelo declarativo (e.g., Capítulo 2, Prolog) (1), (2)&(3)	modelo concurrente dirigido por los datos (e.g., Sección 4.1) (1), (2)&(3)
ejecución perezosa	programación funcional perezosa (e.g., Haskell) (1)&(2), (3)	PF perezosa con vars. de flujo de datos (1), (2), (3)	modelo concurrente dirigido por demanda (e.g., Sección 4.5.1) (1), (2), (3)

(1): Declarar una variable en el almacén
(2): Especificar la función para calcular el valor de la variable
(3): Evaluar la función y ligarla a la variable

(1)&(2)&(3): Declaración, especificación, y evaluación coinciden
(1)&(2), (3): Declaración y especificación coinciden; evaluación se realiza después
(1), (2)&(3): Declaración se hace primero; especificación y evaluación se hacen después y coinciden
(1), (2), (3): Declaración, especificación, y evaluación se realizan por separado