ANEXO C

Sintaxe e Semântica da Linguagem Esterel

C.1 Módulos e submódulos

Os programas Esterel são estruturados a partir das suas unidades básicas, os módulos. Um módulo tem um nome, uma declaração de interface e um corpo que é executável.

Module <nome> :
<declaração de interface>
<corpo>
end module

Um módulo pode utilizar submódulos que são módulos instanciados pela construção "**run**"; não pode haver recursividade sobre a instanciação.

C.2 Declaração de interface

A declaração de interface define os objetos que um módulo importa ou exporta. Ela contém *objetos de dados* declarados de forma abstrata em Esterel e implementados externamente e *objetos de interface reativa*.

C.2.1 Dados

As declarações de dados declaram os objetos que manipulam dados:

- **Tipos e operadores**

Os cinco tipos primitivos de Esterel são: **boolean**, **integer**, **float**, **double** e **string**. As operações são as usuais: igualdade "=" e diferença "< >" para todos os tipos, "**and**", "**or**" e "**not**" para o tipo **boolean** e "**+, -, *, /, <, < =, >, >=**" para tipos **integer**, **float**, **double**. O usuário pode definir seus próprios tipos declarando seus nomes; um tipo do

usuário é um objeto abstrato cuja definição será dada somente na linguagem hospedeira.

- **Constantes**

É possível declarar constantes de qualquer tipo. Quando o tipo é pré-definido, são declarados em Esterel nome, tipo e valor; quando o tipo é definido pelo usuário são apenas declarados nome e tipo, sendo que o valor é definido na linguagem hospedeira.

- **Funções**

A declaração de função contém a lista de tipos dos objetos que vai usar e o tipo do objeto de retorno: "**function <nome> (<lista de tipo de argumentos>) : <tipo do retorno>;**".

- **Procedimentos**

A declaração de um procedimento tem duas listas (opcionais) de argumentos de tipos arbitrários: lista de argumentos passados por referência e modificáveis pela chamada ("**call**"), lista de argumentos passados por valores e não modificáveis. A declaração se apresenta na forma: "**procedure <nome> (<lista de tipo de argumento-referência>) (<lista de tipo de argumento-valor>);**".

Procedimentos e funções são definidos na linguagem hospedeira e não apresentam efeitos colaterais.

- **Tarefas**

As tarefas são entidades de cálculo externo mas que não podem ser consideradas instantâneas. Elas são sintaticamente similares aos procedimentos e são executadas pela construção "**exec**" acoplada com o sinal "**return**" (ver a seguir)

C.2.2 Sinais e Sensores

Os sinais e sensores constituem a interface reativa do módulo. Os sinais são difundidos instantaneamente em todo o programa. O valor difundido de um sinal com valor ou de um sensor é único a cada instante.

- **Declaração de sinais de interface**

Os sinais de interface são de entrada "**input**", de saída "**output**", de entrada-saída "**inputoutput**" e de terminação de tarefas externas "**return**". Os sinais se dividem em sinais puros (por exemplo "**input <nome-sinal>;**") que tem apenas um estado de presença ("*presente*" ou "*ausente*") e sinais com valor que transportam também um

valor de tipo arbitrário (por exemplo "**output <nome-sinal> := <valor inicial> : <tipo-sinal>;**").

Um sinal com valor não pode ser emitido pelo programa duas vezes no mesmo instante e nem no mesmo instante que ele esta sendo recebido do ambiente. Ele é chamado de sinal com valor único. Para se livrar desta restrição, utiliza-se a palavra chave "**combine**" na declaração do sinal com valor; os sinais são chamado de sinais com valor combinados. Operadores (and, or, +, *) e outras funções declarados pelo usuário podem ser usados na combinação de sinais.

Existe apenas um sinal puro pré-definido "**tick**" que representa o relógio de ativação do programa reativo mas não precisa declara-lo. Seu estado tem o valor "*presente*" a cada instante.

- **Sensores**

Os sensores são sinais de entrada com valor mas sem a presença da informação de estado: "**sensor <nome-sensor> : <tipo-sensor>;**". O valor do sensor é acessado pelo programa através da construção "**?**", quando necessário

- **Relações de entrada**

A construção "**relation ...**" permite representar relações de entrada que indicam condições booleanas entre os sinais "**input**" e "**return**"; estas condições são supostamente garantidas pelo ambiente. As relações são de incompatibilidade ou exclusão entre os sinais ("#") ou de sincronização entre sinais ("=>").

- **Declaração de sinal local**

Sinais podem ainda ser declarados localmente pela declaração "**signal <lista de sinais> in p end signal**" sem aparecer na interface do módulo. A declaração dos sinais locais é feita da mesma forma que a dos sinais de interface e o escopo dos sinais locais é o corpo da construção **p**. Numa malha, um sinal local pode ser executado várias vezes no mesmo instante, criando a cada execução uma nova copia.

C.2.3 Variáveis

As variáveis são objetos aos quais valores podem ser atribuídos. A declaração das variáveis com seu nome, valor inicial e tipo é feita numa construção de declaração de variável local "**var <lista de variáveis> in p end var**". O escopo da declaração de variável e o corpo da construção **p**. A modificação da variável pode ser o resultado de atribuições, chamadas de procedimentos e execuções de tarefas externas ("**exec**"). Contrariamente ao sinal, uma variável pode tomar várias valores sucessivos no mesmo instante.

C.2.4 Expressões

As expressões possíveis em Esterel são:

- **Expressões de dados**

Elas combinam constantes, variáveis, sensores e sinais com valores usando operadores e chamadas de função. A sua avaliação é instantânea e envolve checagem de tipos. "**?S**" indica o valor corrente do sensor ou do sinal com valor **S**.

- **Expressões de sinais**

Elas são expressões booleanas ("**not**", "**and**", e "**or**") aplicadas sobre os estado de sinais (ou do sinal "tick"). Elas são utilizados em testes de presença ou em expressões de atraso.

- **Expressões de atraso**

Elas são utilizadas em construções temporais tais como "**await**" ou "**abort**" para expressar atrasos que começam quando inicia a construção temporal que as contém. Existem três tipos possíveis:

- atrasos padrões definidos por expressões de sinais e que nunca esgotam instantaneamente;
- atrasos imediatos definidos por "**immediate [<expressão de sinais>]**" e que esgotam instantaneamente;
- atrasos de contagem definidos por uma expressão de contagem inteira seguida por uma expressão de sinais: ("**<expressão-contagem> [<expressão-sinais>]**").

C.3 Construções do corpo

Todas as construções utilizáveis no corpo do módulo são descritas a seguir, a exceção da declaração do sinal local já descrito anteriormente.

- **Construções básicas de controle**

As construções básicas de controle são "**nothing**" que termina instantaneamente, "**pause**" que para e termina no próximo instante , "**halt**" que para para sempre sem nunca terminar.

- **Atribuição**

A atribuição "**X := e**" com a variável **X** e a expressão de dados **e** do mesmo tipo é instantânea.

- **Chamada de procedimento**

A chamada de procedimento tem a forma "**call P (X, Y) (e_1, e_2)**" onde **X**, **Y** são variáveis e $\mathbf{e_i}$ são expressões. A chamada é instantânea.

- **Emissão de sinal**

A emissão instantânea de sinal é realizada por "**emit S**" ou "**emit S(e)**" respectivamente no caso de um sinal puro ou de um sinal com valor resultante da avaliação da expressão **e**. Para um sinal único, somente um "**emit**" pode ser executado neste instante; para um sinal combinado, o valor emitido é combinado com os que são emitidos por outros "**emit**" neste instante, usando a função de combinação.

A emissão contínua de um sinal é realizada pela construção "**sustain S**" ou "**sustain S(e)**" que fica ativa para sempre e emite **S** ou **S(e**) a cada instante.

- **Seqüência**

O operador de seqüência "**;**" permite que a construção **q** de "**p; q**" inicia imediatamente após a construção **p** ter terminada, a menos que **p** contenha algum "**exit**" de um "**trap**".

- **Malha**

A construção "**loop p end loop**" representa a malha simples infinita. O corpo **p** é sempre re-iniciado imediatamente após seu término. Se construções "**trap**" fazem parte do corpo **p**, os "**exit**" são propagados instantaneamente e a malha para. Não é permitido que o corpo **p** de uma malha possa terminar instantaneamente quando iniciado.

A malha repetitiva executa seu corpo **p** um número finito de vezes. Não é permitido que o corpo **p** termina instantaneamente. A construção "**repeat e times p end repeat**" na qual **e** é do tipo "**integer**" e é avaliada somente uma vez no instante inicial. Se o resultado da avaliação for zero ou negativo, o corpo não será executado. A construção "**repeat**" é considerada como instantânea e não poderá ser colocada numa malha "**loop**" se não for precedido ou seguido por um atraso. Para garantir, neste caso, que o corpo será executado pelo menos uma vez, utiliza-se a construção "**positive repeat ...**" que permite realizar o teste para repetição somente após a primeira execução do corpo.

- **Testes**

O teste de presença binário tem a seguinte forma geral "**present e then p else q end present**"; uma das ramificações "**then**" ou "**else**" pode ser omitida e neste caso a omitida tem o significado de "**nothing**". O teste de presença múltiplo utiliza a construção "**case**" dentro do "**present**" da forma seguinte:

present
case e_1 **do** p_1
case e_2 **do** p_2
case e_3 **do** p_3
else q
end present

O teste condicional utiliza a construção "**if**" para testar expressões de dados booleanas. O teste condicional binário tem a seguinte forma "**if <condição> then p else q end if**". O teste condicional múltiplo não usa o "**case**" reservado para os teste de presença de sinal mas pode ser realizado em seqüência usando a palavra chave "**elseif**" da forma seguinte:

if **<condição 1> then** p_1
elseif <condição 2> then p_2
elseif <condição 3> then p_3
else q
end if

- **Atraso**

A construção temporal mais simples "**await S**" significa a espera por um atraso. Quando a construção é iniciada, ela entra em pausa até o atraso ser esgotado, instante no qual ela termina.

A construção "**await immediate S**" termina instantaneamente se o sinal for presente ou a expressão de sinal for verdadeira no instante inicial.

Pode se utilizar ainda a construção "**case**" dentro do "**await**"; o primeiro atraso que se esgota fixa o comportamento seguinte; se dois deles se esgotam simultaneamente, a prioridade é com o primeiro da lista; a cláusula "**else**" não é permitida. A contagem do número de sinais ou uma expressão de sinal pode também ser utilizada para expressar o atraso.

A construção "**await**" completa utiliza uma cláusula "**do**" para iniciar outra construção quando o atraso esgota: "**await <expressão-atraso> do p end await**".

- **Preempção**

A construção "**abort p when <expressão-atraso> do q**" realiza uma preempção

forte do corpo **p** mas não entrega o controle a este no instante de preempção.

A construção "**weak abort p when <expressão-atraso> do q**" realiza uma preempção fraca na qual o corpo **p** recebe o controle para um último instante no momento de preempção.

A claúsula "**do**" permite executar uma construção **q** se o atraso esgotar, imediatamente no caso de "**abort**" e após o último instante no caso de "**weak abort**".

A introdução da palavra chave "**immediate**" nestas construções na forma "**... when immediate <expressão-atraso> ...",** permite ao atraso de se esgotar imediatamente no instante de inicio se a expressão de atraso for verificada neste; o corpo **p** da construção não se executa no caso "**abort**" e se executa para um último instante no caso "**weak abort**".

Uma lista de "**case**" pode também ser introduzida nestas construções de preempção. Construções "**abort**" aninhadas são também possíveis e estabelecem prioridades; por exemplo J é prioritário sobre I se os dois ocorrem simultaneamente e q não será iniciado neste caso:

```
abort
    abort p
    when I do q
    end abort
when J
```

A construção "**suspend p when <expressão-sinal>**" tem um efeito de suspensão (do tipo ^Z do Unix). O corpo **p** é imediatamente iniciado quando a construção "**suspend**" inicia,. A cada instante, se a expressão de sinal for "*true*", o corpo **p** se suspende no seu estado atual e a construção "**suspend**" faz uma pausa neste instante; se a expressão de sinal for "*false*", o corpo **p** é executado neste instante. Para realizar o teste da expressão de sinal no primeiro instante é necessário utilizar "**suspend p when immediate <expressão-sinal>**".

- ## Malha temporal

As malhas temporais são infinitas e o único meio de termina-las é através de uma exceção ("**exit**" de um "**trap**"). Existem duas formas de declarar malhas temporais:

- "**loop p each d**" onde d é um atraso não imediato. O corpo **p** é inicializado ao instante inicial, e re-inicializado a cada vez que o atraso **d** esgota; se **p** termina antes do esgotamento de **d**, o re-inicio de **p** deverá esperar até o atraso **d** se esgotar. Esta construção é uma abreviação de:

```
loop
        abort
                p; halt
        when d
end loop
```

- "**every d do p end**" que difere do anterior pela espera inicial de **d** antes de iniciar o corpo **p**. Esta construção abrevia:

```
await d;
loop
        p
each d
```

Esta construção pode ser imediata "**every immediate d do p end**" e neste caso, no instante inicial, **p** inicia imediatamente se o atraso **d** esgota imediatamente.

- **Exceção e tratamento**

O mecanismo de exceção é implementado pela construção:

```
trap T in
        p
handle T do
        q
end trap
```

O corpo **p** inicia imediatamente quando a construção "**trap**" inicia. A sua execução continuará até o término de **p** ou a saída através da execução da construção "**exit T**" que leva o "**trap**" a terminar imediatamente, abortando **p** por preempção fraca. O tratamento da exceção (quando houver) é realizado pela construção "**handle**", que permite o inicio imediato de **q** após corpo **p** ter sido abortado pelo "**exit**" do "**trap**":

Quando se tem construções "**trap**" aninhadas, a mais externa tem a maior prioridade. Várias exceções podem ser declaradas numa única construção "**trap**"; todas essas exceções são concorrentes e tem o mesmo nível de prioridade; no caso de várias exceções ocorrer simultaneamente, seus tratadores de exceção serão executados em paralelo.

As construções "**trap**" podem ter valores como os sinais. Inicialização de valor e "**trap**" combinados são permitidos como no caso dos sinais. A passagem de um valor ao tratador de exceção é possível e é o resultado da expressão "**??S**" que se encontra somente neste tratador.

- **Paralelismo**

O operador de paralelismo "**ll**" (que pode ter qualquer aridade) coloca as construções que ele separa em paralelismo síncrono. Os sinais emitido por um dos ramos ou por outra parte do programa são difundidos instantaneamente a todas os ramos. Somente variáveis para leitura podem ser compartilhadas em todos os ramos da construção paralela.

Uma construção paralela, quando inicia, dispara instantaneamente uma "*thread*" por ramo. Ela terminará quando todos os ramos terão terminado, esperando eventualmente até o último terminar. Se o "**exit**" de uma exceção "**trap**" é ativada num ramo, ele é propagada em todos os outros ramos, levando a uma preempção fraca ("**weak abort**") de todos os ramos no mesmo tempo.

C.4 Instanciação de módulo

A instanciação de um módulo dentro de outro módulo é possível a partir da construção executável "**run <nome-módulo>**".A instanciação recursiva de submódulos é proibida.

Como todos os dados são globais em Esterel, as declarações de dados de um submódulo instanciado são exportados no módulo pai e vice-versa. As declarações de interface de sinais e de relações devem ser descartadas; as interface de sinais de um submódulo instanciado deve existir no módulo pai com o mesmo tipo.

A renomeação de objeto de interface é possível em tempo de instanciação de módulo conforme definido na construção "**run <nome-módulo> [X / Y; ...]**" que permite a renomeação de Y por X, desde que os tipos e operadores de tipos coincidam ("match"). O objeto renomeando X pode ser uma constante ou um operador ou um identificador; o objeto renomeado Y deve ser um identificador pertencente a interface de sinal ou a dados de módulo instanciado.

C.5 A execução de tarefa externa

O controle da execução de tarefas externas que levam tempo usa o mecanismo "**exec**". Essas tarefas se comportam como procedimentos a serem executados assincronamente. O programa Esterel se interessa apenas pelo inicio e pelo fim delas ou pela suspensão ou preempção das mesmas por outras construções Esterel.

A execução de uma tarefa é realizada por:

> "**exec Task (<parâmetros-referência>) (<parâmetros-valores>) return R**"

na qual **R** é o sinal de retorno. Quando deseja-se, simultaneamente, controlar várias tarefas, utiliza-se:

exec
case T_1 (...) (...) return R_1 do p_1
...
case T_n (...) (...) return R_n do p_n
end exec