5

Traduções

Neste capítulo serão apresentadas quatro traduções: de jogo sem estratégias em *estruturas de Kripke* (seção 5.1.1); de jogo com estratégias em *estruturas de Kripke* (seção 5.1.2); de jogo sem estratégias na linguagem RollGame na linguagem do verificador de modelos (seção 5.2.1); de jogo com estratégias na linguagem RollGame na linguagem do verificador de modelos (seção 5.2.2). E, por fim, na seção 5.3 será apresentado uma ferramenta que automatizará as traduções de jogos na linguagem RollGame na linguagem do verificador de modelos.

5.1 Traduções da Definição de Jogo Proposta em *Estrutura de Kripke*

Como há duas definições de jogo, com estratégias e sem estratégias, serão apresentadas duas traduções de jogo em *estruturas de Kripke*. Assim, pode-se entender jogo em modelo lógico. As traduções dão-se de forma quase direta.

Definição 23 *Estrutura de Kripke* ($\mu = (S, S_o, R, L)$) :

- *um conjunto de estados* S;
- *um conjunto de estados iniciais* S_o, *onde* $S_o \subseteq S$;
- *uma relação de transição* $R \subseteq S \times S$;
- *uma função de rótulos* $L : S \to \wp(\mathcal{V})$, *onde* $\mathcal{V}$ *é o conjunto de proposições atomicas.*

5.1.1 Tradução de Jogo Sem Estratégias em *Estrutura de Kripke*

Definição 24 *Tradução de jogo sem estratégias em estrutura de Kripke:*

Sejam $\mathcal{G} = (J, \mathcal{SE}, e_o, \mathcal{CA})$ *um jogo sem estratégias, cuja linguagem do jogo é dada por* $\mathcal{L}_\mathcal{G} = \bigcup_{\substack{\forall e \in \mathcal{SE} \mid \\ e=(J', val_1, ..., val_m)}} \{J \approx J', var_1 \approx val_1, ..., var_m \approx val_m\}$,[1] *e* $\mu = (S, S_o, R, L)$ *uma estrutura de Kripke, uma tradução de* $\mathcal{G}$ *em* μ *é uma aplicação* $T : \mathcal{G} \rightarrow \mu$ *tal que:*

- $S = \mathcal{SE}$;
- $S_o = e_o$;
- $R = \mathcal{CA}$;
- $L(s) = \{J \approx J', var_1 \approx val_1, ..., var_m \approx val_m \mid \forall s \in S \text{ onde } s = (J', val_1, ..., val_m)\}$.

A figura 5.1.1.1 mostra a *estrutura de Kripke* associado a tradução do jogo sem estratégias (figura 4.1.1).

[1] Os termos da linguagem são proposições. Por exemplo, $var_1 \approx var_2$ não significa que var_1 é igual a var_2 e sim uma proposição denominada $var_1 \approx var_2$.

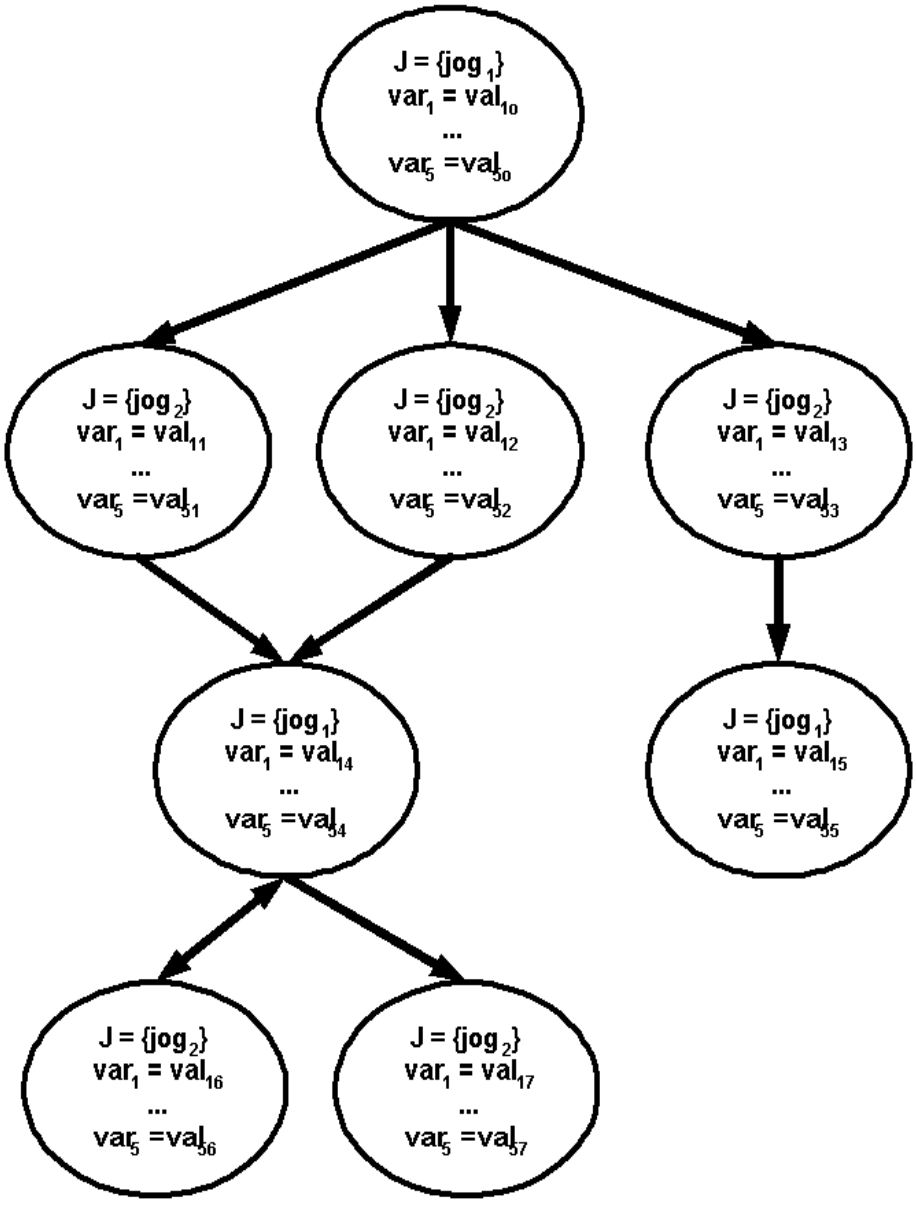


Figura 5.1.1.1 - Estrutura de Kripke gerada a partir da tradução da definição de jogo sem estratégia em estrutura de Kripke.

5.1.2 Tradução de Jogo Com Estratégias em *Estrutura de Kripke*

Definição 25 *Tradução de jogo com estratégias em estrutura de Kripke*

Dados $\mathcal{GE}= (J, \mathcal{SE}, e_o, \mathcal{CA}, \mathcal{CE})$ *um jogo com estratégias, cuja linguagem do jogo é dada por* $\mathcal{L}_\mathcal{G} = \bigcup_{\substack{\forall e \in \mathcal{SE} \mid \\ e=(J', val_1, ..., val_m)}} \{J \approx J', var_1 \approx val_1, ..., var_m \approx val_m\}$,[2] *e* $\mu = (S, S_o, R, L)$ *uma estrutura de Kripke, uma tradução de* $\mathcal{GE}$

[2]Os termos da linguagem são proposições. Por exemplo, $var_1 \approx var_2$ não significa que

em μ *é uma aplicação* $T : \mathcal{G}E \to \mu$ *tal que:*

- $S = \mathcal{S}E$;
- $S_o = e_o$;
- $R = \bigcup_{\substack{\forall <e\mathcal{A}e',\alpha> \in \mathcal{C}E \text{ tal que} \\ ||<e\mathcal{A}e',\alpha>||_{\mu\mathcal{G},e}=e'}} \mathcal{A}$;
- $L(s) = \{J \approx J', var_1 \approx val_1, ..., var_m \approx val_m \mid \forall s \in S \text{ onde } s = (J', val_1, ..., val_m)\}$.

A figura 5.1.2.1 mostra a *estrutura Kripke* associado a tradução do jogo com estratégias (figura 4.3.1).

---

var_1 é igual a var_2 e sim uma proposição denominada $var_1 \approx var_2$.

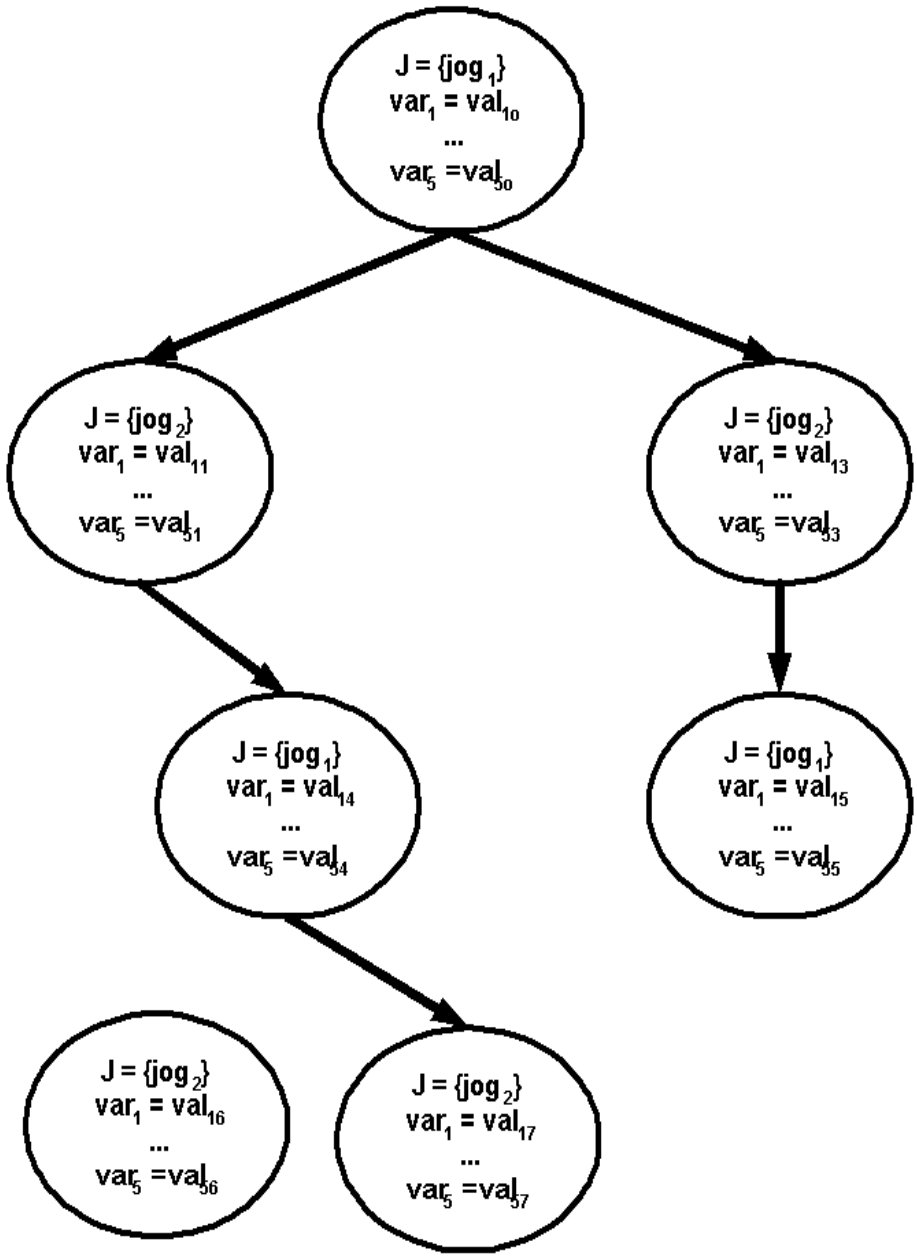


Figura 5.1.2.1 - Estrutura de Kripke gerada a partir da tradução da definição de jogo com estratégia em estrutura de Kripke.

5.2 Traduções da linguagem RollGame na Linguagem do Verificador de Modelos

A linguagem de RollGame possui as duas definições de jogos, com e sem estratégias. Como descrito na seção 4.5 a distinção é feita pela existência ou não da palavra reservada "*ESTRATEGIAS*". Portanto, deve-se definir duas traduções na linguagem do verificador de modelos: uma para jogo em RollGame sem estratégias e outra para jogo em RollGame com estratégias.

Lembre-se que o verificador de modelos escolhido foi o SMV (Symbolic Model Verifier). Assim, a tradução será feita na linguagem de especificação de modelos do SMV.

A linguagem RollGame foi modificada devido as limitações da linguagem de especificação de modelos do SMV, pois somente os domínios da linguagem do SMV foram utilizados.

Uma outra modificação em RollGame será a inserção das especificações em CTL na linguagem do SMV. Estas serão utilizadas para a construção e análise de estratégias em um jogo[3]. A BNF da linguagem modificada é mostrada na figura 5.2.1.

[3]No capítulo 6 será explicado como se fazer a construção e análise de estratégias em jogos.

<jogo> :==
ESTADO <*estado*> ESTADO_INICIAL <*estado_inicial*>
ACOES <*acao*>* DEFINICOES <*definicao*>*
| ESTADO <*estado*> ESTADO_INICIAL <*estado_inicial*>
ACOES <*acao*>* ESTRATEGIAS <*estrategia*>* DEFINICOES <*definicao*>*
<**estado**> :==
VARIAVEIS_ESTADO <def_variaveis>* JOGADORES <def_jogador>*
<**estado_inicial**> :==
VARIAVEIS_ESTADO_INICIAIS <*atribuicao*>*
JOGADORES_INICIAIS <*atribuicao_booleana*>*
<**definicao**> :==
<*id*> := <sentenca> ;
<**def_variaveis**> :==
<*id*> : <*variaveis*> ;
<**def_jogador**> :==
<*id*> : boolean ;
<**variaveis**> :==
boolean
| {<*tipo_enumerado*> }
<**tipo_enumerado**> :==
<*tipo_dado*> , < *tipo_enumerado*>
| <*tipo_dado*>
<**sentenca**> :==
(<*tipo_dado*> = <*tipo_dado*>)
| (! <*sentenca*>)
| (<*sentenca*> <*operadores*> <*sentenca*>)
<**operadores**> :==
<*operadores_logicos*>
| <*operadores_matematicos*>
<**operadores_logicos**> :==
& | | | -> | < -> | =
<**operadores_matematicos**> :==
< = | >= | > | < | + | - | * | / | mod
<**acao**> :==
<*sentenca*> -> { <*atribuicao*>* } ;
<**formula_gal**> :==
(<tipo_dado> = <tipo_dado>)
| (! <formula_gal>)
| (<formula_gal> <operadores> <formula_gal>)
| EX(<formula_gal>) | AX(<formula_gal>)
| EF(<formula_gal>) | AG(<formula_gal>)
| EG(<formula_gal>) | AF(<formula_gal>)
| E(<formula_gal> U <formula_gal>)
| A(<formula_gal> U <formula_gal>)
<**estrategia**> :==
<*formula_gal*> -> {<*atribuicao*>*}
<**atribuicao**> :==
<*id*> := <*tipo_dado*> ;
| <id> := { <*tipo_enumerado*> } ;
<***tipo_dado***> :==
<*id*>
| <*number*>
<**atribuicao_booleana**> :==
<*id*> := <*valores_booleanos*> ;
<**valores_booleanos**> :==
0 | 1 | true | false

Figura 5.2.1- BNF da linguagem *RollGame modificada*

5.2.1 Tradução de Jogo Sem Estratégia em RollGame na Linguagem do SMV

A tradução estará apresentada de forma esquemática (figura 5.2.1.1) e será feita a partir de um jogo sem estratégias em RollGame (figura 4.5.2) da seguinte forma:

1. Será construído um módulo estado (*MODULE ESTADO*) que irá definir o conjunto de estados, o estado inicial e as possíveis definições utilizadas em RollGame.

2. Para cada ação ($acao_i$) definida em RollGame, será criado um módulo ação ($MODULE\ ACAO_i$), passando como parâmetro a instância do módulo estado, com uma condição de *fairness* para garantir que a ação poderá ser aplicada. Nestes módulos estarão definidas as transições das variáveis de estados e as transições de jogadores. As variáveis e definições dentro de cada módulo ação serão precedidas de "e.".

3. Por fim, será criado um módulo (*MODULE main*) que irá instanciar o estado e as estratégias. Os módulos serão compostos usando *interleaving*, pois cada estratégia será aplicada separadamente.

A tradução do exemplo de um jogo sem estratégias em RollGame (Figura 4.5.4) terá como semântica esperada da linguagem do SMV a *estrutura de Kripke* apresentada na figura 5.2.1.2. Pode-se observar que a semântica apresentada difere da tradução da definição de jogo sem estratégias (figura 5.1.1.1) por dois motivos:

1. Em *estruturas de Kripke* da semântica de SMV não existem estados terminais, contudo pode-se caracterizar o término de um jogo não só pela inexistência de uma relação com outro estado, mas, também, através de uma condição de término de jogo. Assim, consegue-se caracterizar quando um jogo chega ao seu término.

2. Os estados não-terminais têm auto-relações (linhas pontilhadas), pois na tradução de jogo em RollGame cada ação é mapeada em um módulo ação e os módulos são compostos através de *interleaving* com condições de *fairness running*. Logo, as auto-relações serão limitadas e não existirá um caminho no qual as auto-relações ocorram indefinidamente. Desta forma, o comportamento do jogo é equivalente ao da tradução a menos de algumas auto-relações.[4]

[4]O conceito de *interleaving* e de *fairness running* foram apresentado na seção 2.2.1

MODULE ESTADO e
VAR
 var_1 : $type_1$;
 ... ;
 var_M : $type_M$;
 $jogador_1$: boolean;
 ... ;
 $jogador_{|J|}$: boolean;
ASSIGN
 var_1 := var_{1o};
 ... ;
 var_M := var_{Mo};
 $jogador_1$:= $jogador_{1o}$;
 ... ;
 $jogador_{|J|}$:= $jogador_{|J|o}$;
DEFINE
 $definicao_1$:= $condicao_1$;
 ... ;
 $definicao_P$:= $condicao_P$;

MODULE ACAO$_1$(e)
ASSIGN
next($atribuicao_1$(var)**) :=**
 case
  $sentenca_1$: $atribuicao_1$(valor);
  1 : $atribuicao_1$(var);
 esac;
...
next($atribuicao_W$(var)**) :=**
 case
  $sentenca_1$: $atribuicao_W$(valor);
  1 : $atribuicao_W$(var);
 esac;
FAIRNESS
 running

MODULE ...

MODULE ACAO$_N$(e)
ASSIGN
next($atribuicao_1$(var)**)** :=
 case
  $sentenca_N$: $atribuicao_1$(valor);
  1 : $atribuicao_1$(var);
 esac;
...
next($atribuicao_K$(var)**)** :=
 case
  $sentenca_N$: $atribuicao_K$(valor);
  1 : $atribuicao_K$(var);
 esac;
FAIRNESS
 running

MODULE main
VAR
 e : ESTADO;
 $acao_1$: process ACAO$_1$(e);
 ... ;
 $acao_N$: process ACAO$_N$(e);

Figura 5.2.1.1 - Esquema da tradução de um jogo sem estratégia em RollGame na lingugem do SMV

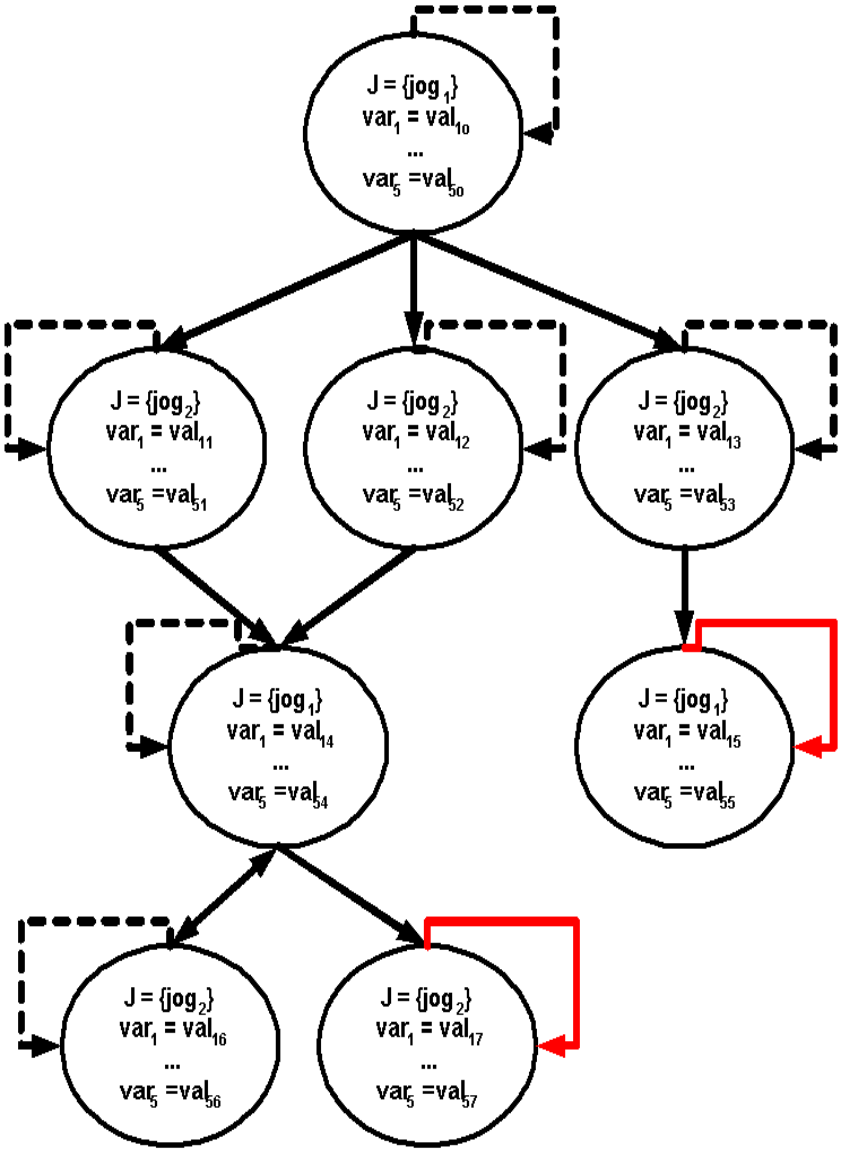


Figura 5.2.1.2 - Estrutura de Kripke gerada a partir da tradução do jogo em RollGame na linguagem do verificador de modelos.

5.2.2 Tradução de Jogo Com Estratégias em RollGame na Linguagem do SMV

A tradução de jogo com estratégias em RollGame na linguagem do verificador de modelos é semelhante a tradução de jogo sem estratégias. Porém, duas diferenças devem ser observadas em particular: a primeira e mais simples, é que a análise do jogo deve ser realizada nas estratégias e não nas ações; a segunda é que os condicionais das estratégias podem ser temporais e serão verdadeiros ou não nas ações, e não nas estratégias.

Contudo, definir condicionais temporais na linguagem de especificação do verificador de modelos é um problema, pois não se conhece nenhum verificador

de modelos que utilize nos condicionais de transições fórmulas temporais. A seguir, duas possíveis soluções para este problema:

1. Desenvolver um verificador de modelos que na sua linguagem de especificação tivesse fórmulas de GAL nas sentenças de transições. Esta solução não foi adotada, uma vez que, aparentemente, é uma tarefa não trivial. Além disso, um motivo mais importante do que este é que a solução ficaria necessariamente delimitada àquele verificador de modelos desenvolvido;

2. Traduzir as fórmulas de GAL em sentenças de transições de estados que representassem as fórmulas em GAL. Para tanto, é necessário embutir nas sentenças a estrutura de um jogo. Se uma fórmula de GAL for uma sentença de transição de estados, então não é preciso realizar a tradução; esta solução foi adotada com sucesso e pode ser utilizada em vários verificadores de modelos. Esta abordagem só é válida para jogos finitos. A seguir, a definição das fórmulas em GAL.

Definição 26 *Sentença de transição de estados de um jogo ($\digamma$):*
Sejam α e β jogadores ou termos de um sorte $\mathcal{S}$.

- $\digamma :== true \mid false \mid jog \mid \alpha = \beta \mid (\neg \digamma_1) \mid (\digamma_1 \vee \digamma_2) \mid (\digamma_1 \wedge \digamma_2) \mid (\digamma_1 \rightarrow \digamma_2)$;

Notação 1 *Seja $e = e'$ uma notação para $jog_1 = jog_1{}' \wedge \ldots \wedge jog_K = jog_K{}' \wedge val_1 = val_1{}' \wedge \ldots \wedge val_M = val_M{}'$, onde $e = (\{jog_1, ..., jog_K\}, val_1, ..., val_M)$ e $e' = (\{jog_1{}', ..., jog_K{}'\}, val_1{}', ..., val_M{}')$.*

Para definir formalmente a tradução das fórmulas de estratégias de jogos será necessário fazer as seguintes definições: conjunto que contém todos os conjuntos de estados de todos os caminhos ($e\pi$) a partir de um estado; conjunto de estados entre dois estados dado um conjunto de estados de um caminho ($e\Phi$).

Definição 27 *Conjunto que contém todos os conjuntos de estados de todos os caminhos ($e\pi$):*

- $e\pi(e) = e\pi(e, \{e\})$;
- $$e\pi(e,T) = \begin{cases} T, \text{ se } \neg\exists e' \in \mathcal{SE} \textit{ tal que } \exists\ e\mathcal{A}e' \in \mathcal{CA} \text{ e } e' \notin T \\ \bigcup\limits_{\substack{\forall e\mathcal{A}e' \in \mathcal{CA} \\ tal\ que\ e' \notin T}} e\pi(e', T \cup \{e'\}), \text{ caso contrário} \end{cases}$$

Definição 28 *Conjunto de estados entre dois estados ($e\Phi$) dados um conjunto de estados de um caminho Ec e dois estados e_α,$e_\beta \in Ec$:*

- $e\Phi(e_\alpha, Ec, e_\beta) = \begin{cases} \{e_\alpha\} \bigcup e\Phi(e', Ec - \{e_\alpha\}, e_\beta), \text{ se } e_\alpha \neq e_\beta \text{ e } \exists e_\alpha \mathcal{A} e' \in \mathcal{CA} \\ \\ \oslash, \text{ caso contrário} \end{cases}$

Definição 29 *Tradução das fórmulas de GAL em sentenças de transições de estados*[5]*:*

Sejam um jogo $\mathcal{G} = (J, \mathcal{SE}, e_o, \mathcal{CA})$ *e um estado* e*. Uma tradução de uma fórmula em GAL em uma sentença de transição de estados é uma aplicação* $\Theta : GAL \times \mathcal{SE} \rightarrow \digamma$*, tal que:*

- $\Theta(jog, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}} (e = e' \rightarrow (\varsigma(e', jog)))$
- $\Theta(P_\mathcal{D}(t_1, ..., t_n), e) \equiv$
  $\bigwedge_{\forall e' \in \mathcal{SE}} (e = e' \rightarrow \gamma)\,, onde\, \gamma = \begin{cases} true, \text{ se } < \bar{\sigma}_\mathcal{D}(e', t_1), ..., \bar{\sigma}_\mathcal{D}(e', t_n) > \in P_{\mathcal{D}_{\mu\mathcal{G}}} \\ false, \text{ caso contrário} \end{cases}$
- $\Theta((\alpha^\mathcal{D} \approx \beta^\mathcal{D}), e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}} (e = e' \rightarrow \gamma)\,, onde\, \gamma = \begin{cases} true, \text{ se } \bar{\sigma}_\mathcal{D}(e', \alpha^\mathcal{D}) = \bar{\sigma}_\mathcal{D}(e', \beta^\mathcal{D}) \\ false, \text{ caso contrário} \end{cases}$
- $\Theta((\neg\alpha), e) \equiv (\neg\Theta(\alpha, e))$
- $\Theta((\alpha \vee \beta), e) \equiv (\Theta(\alpha, e) \vee \Theta(\beta, e))$
- $\Theta((\alpha \wedge \beta), e) \equiv (\Theta(\alpha, e) \wedge \Theta(\beta, e))$
- $\Theta((\alpha \rightarrow \beta), e) \equiv (\Theta(\alpha, e) \rightarrow \Theta(\beta, e))$
- $\Theta([\exists\bigcirc]\alpha, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e'\mathcal{A}s]} \left(e = e' \rightarrow \left(\bigvee_{\forall \mathcal{A} \in \mathcal{CA}, \forall e'' \in \mathcal{SE} \mid e'\mathcal{A}e''} \Theta(\alpha, e'') \right) \right) \bigwedge$
  $\bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e'\mathcal{A}s]} (e = e' \rightarrow false)$
- $\Theta([\forall\bigcirc]\alpha, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e'\mathcal{A}s]} \left(e = e' \rightarrow \left(\bigwedge_{\forall \mathcal{A} \in \mathcal{CA}, \forall e'' \in \mathcal{SE} \mid e'\mathcal{A}e''} \Theta(\alpha, e'') \right) \right) \bigwedge$
  $\bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e'\mathcal{A}s]} (e = e' \rightarrow false)$
- $\Theta([\exists\Box]\alpha, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e'\mathcal{A}s]} \left(e = e' \rightarrow \left(\bigvee_{\forall C \,\in\, e\pi(e')} \left(\bigwedge_{\forall e'' \in C} \Theta(\alpha, e'') \right) \right) \right) \bigwedge$
  $\bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e'\mathcal{A}s]} (e = e' \rightarrow false)$

[5]Lembre-se que o jogo é finito, isto é, o conjunto de ações e o conjunto de estados são finitos. A função $\varsigma(e, jog)$ e a função $\bar{\sigma}_\mathcal{D}(e, t)$ são definidas na seção 4.2.

- $\Theta([\exists\diamond]\alpha, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e' \mathcal{A} s]} \left(e = e' \rightarrow \left(\bigvee_{\forall C \, \in \, e\pi(e')} \left(\bigvee_{\forall e'' \in C} \Theta(\alpha, e'') \right) \right) \right) \bigwedge$
$\bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e' \mathcal{A} s]} (e = e' \rightarrow false)$

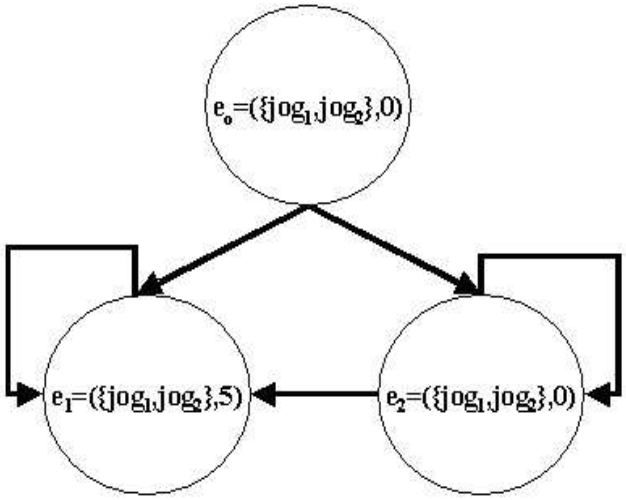
- $\Theta([\forall\Box]\alpha, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e' \mathcal{A} s]} \left(e = e' \rightarrow \left(\bigwedge_{\forall C \, \in \, e\pi(e')} \left(\bigwedge_{\forall e'' \in C} \Theta(\alpha, e'') \right) \right) \right) \bigwedge$
$\bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e' \mathcal{A} s]} (e = e' \rightarrow false)$

- $\Theta([\forall\diamond]\alpha, e) \equiv \bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e' \mathcal{A} s]} \left(e = e' \rightarrow \left(\bigwedge_{\forall C \, \in \, e\pi(e')} \left(\bigvee_{\forall e'' \in C} \Theta(\alpha, e'') \right) \right) \right) \bigwedge$
$\bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e' \mathcal{A} s]} (e = e' \rightarrow false)$

- $\Theta(\exists(\alpha \; \mathcal{U} \; \beta), e) \equiv$
$\bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e' \mathcal{A} s]} \left(e = e' \rightarrow \left(\bigvee_{\forall C \, \in \, e\pi(e')} \left(\bigvee_{\forall e'' \in C} \left(\Theta(\beta, e'') \bigwedge_{\forall e''' \in \, e\Phi(e', C, e'')} \Theta(\alpha, e''') \right) \right) \right) \right)$
$\bigwedge \bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e' \mathcal{A} s]} (e = e' \rightarrow false)$

- $\Theta(\forall(\alpha \; \mathcal{U} \; \beta), e) \equiv$
$\bigwedge_{\forall e' \in \mathcal{SE}[\exists s \in \mathcal{SE}, e' \mathcal{A} s]} \left(e = e' \rightarrow \left(\bigwedge_{\forall C \, \in \, e\pi(e')} \left(\bigvee_{\forall e'' \in C} \left(\Theta(\beta, e'') \bigwedge_{\forall e''' \in \, e\Phi(e', C, e'')} \Theta(\alpha, e''') \right) \right) \right) \right)$
$\bigwedge \bigwedge_{\forall e' \in \mathcal{SE}[\neg\exists s \in \mathcal{SE}, e' \mathcal{A} s]} (e = e' \rightarrow false)$

A correção da tradução de fórmulas em GAL em sentenças de transições de estados é apresentada no apêndice. A figura 5.2.2.1 mostra um exemplo de uma tradução de GAL em sentenças de transição de estados.



$$\Theta([\exists\Box]x=0,e) \equiv \left(\begin{array}{l} \left(e=e_o \rightarrow \left(\begin{array}{l} ((0=0)\wedge(5=0)) \\ \vee((0=0)\wedge(0=0)) \\ \vee((0=0)\wedge(0=0)\wedge(5=0)) \end{array} \right) \right) \\ \bigwedge \left(e=e_1 \rightarrow \left(\ (5=0)\ \right) \right) \\ \bigwedge (e=e_2 \rightarrow ((0=0)\vee((0=0)\wedge(5=0))) \end{array} \right)$$

Figura 5.2.2.1 - Exemplo de tradução de uma fórmula em GAL em sentença de transição de estados

A tradução será apresentada de forma esquemática (figura 5.2.2.2) e será feita a partir de um jogo com estratégia em RollGame (figura 4.5.3) da seguinte forma:

1. Será construído um módulo estado (*MODULE ESTADO*) que irá definir o conjunto de estados, o estado inicial e as possíveis definições utilizadas em RollGame;

2. As fórmulas em *GAL* serão transformados em sentenças de transições de estados como demonstrado anteriormente;

3. Para cada estratégia ($estrategia_i$) definida em RollGame será criado um módulo estratégia ($MODULE\ ESTRATEGIA_i$), passando como parâmetro a instância do módulo estado, com uma condição de *fairness running* para garantir que a estratégia poderá ser aplicada. Nestes módulos estarão definidas as transições das variáveis de estados e as transições de jogadores. As variáveis e definições dentro de cada módulo estratégia serão precedidas de "e.";

4. Por fim, será criado um módulo (*MODULE main)* que irá instanciar o estado e as estratégias. Os módulos serão compostos usando *interleaving*, pois cada estratégia será aplicada separadamente.

A tradução do exemplo de um jogo com estratégias em RollGame (figura 4.5.5) terá como semântica esperada da linguagem do SMV a *estrutura de Kripke* apresentada na figura 5.2.2.3. Como pode-se observar, a semântica apresentada difere da tradução da definição de jogo com estratégias (figura 5.1.1.1) pelos mesmos motivos da semântica apresentada de jogos sem estratégias.

```
MODULE ESTADO e
VAR
  var_1 : type_1;
  ... ;
  var_M : type_M;
  jogador_1 : boolean;
  ... ;
  jogador_|J|: boolean;
ASSIGN
  var_1 := var_1o;
  ... ;
  var_M := var_Mo;
  jogador_1 := jogador_1o;
  ... ;
  jogador_|J| := jogador_|J|o;
DEFINE
  definicao_1 := condicao_1;
  ... ;
  definicao_P := condicao_P;

MODULE ESTRATEGIA_1(e)
ASSIGN
next(atribuicao_1(var)) :=
case
 Θ(formula_gal_1,e)  : atribuicao_1(valor);
 1                   : atribuicao_1(var);
esac;
...
next(atribuicao_W(var)) :=
case
 Θ(formula_gal_1,e)  : atribuicao_W(valor);
 1                   : atribuicao_W (var);
esac;
FAIRNESS
 running
```

```
MODULE ...

MODULE ESTRATEGIA_N(e)
ASSIGN
next(atribuicao_1(var)) :=
case
 Θ(formula_gal_N,e)  : atribuicao_1(valor);
 1                   : atribuicao_1(var);
esac;
...
next(atribuicao_K(var)) :=
case
 Θ(formula_gal_N,e)  : atribuicao_K(valor);
 1                   :atribuicao_K (var);
esac;
FAIRNESS
 running

MODULE main
VAR
 e : ESTADO;
 estrategia_1 : process ESTRATEGIA_1(e);
 ... ;
 estrategia_N : process ESTRATEGIA_N(e);
```

Figura 5.2.2.2 - Esquema da tradução de um jogo com estratégia em RollGame na lingugem do SMV

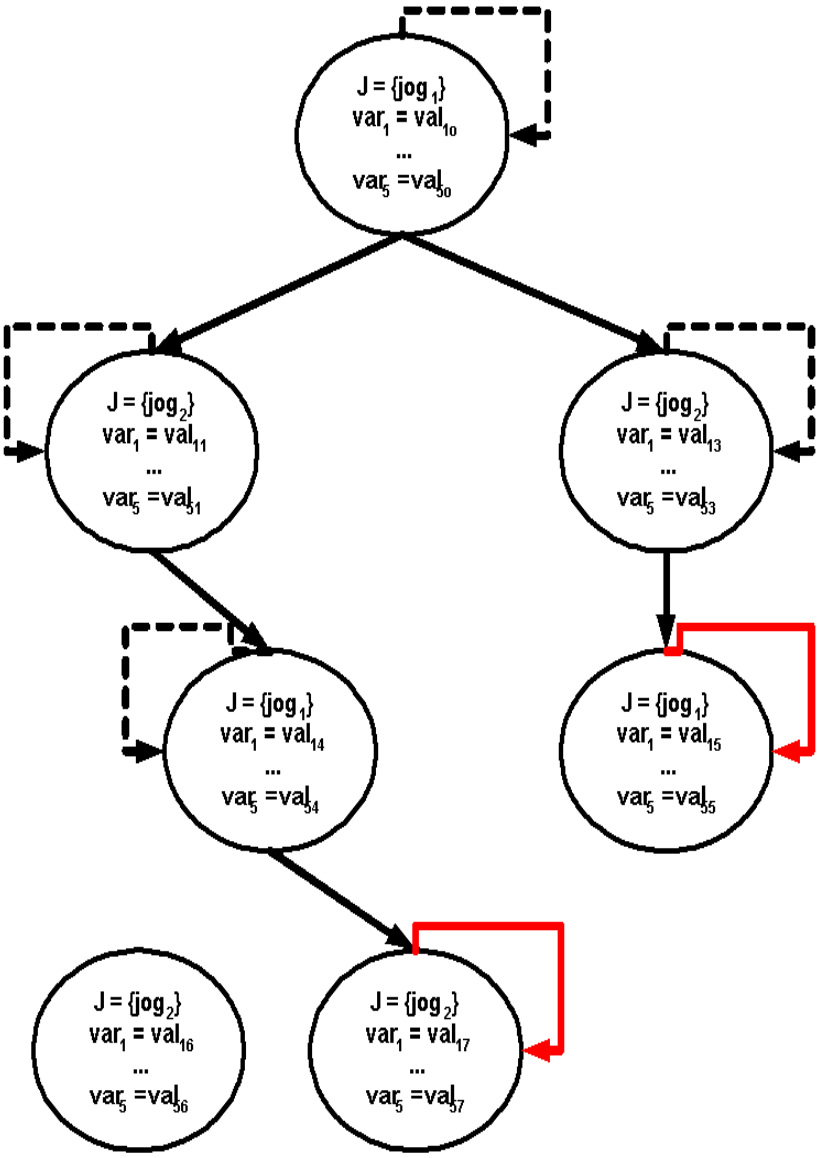


Figura 5.2.2.3 - Estrutura de Kripke gerada a partir da tradução do jogo com estratégia em RollGame na linguagem do verificador de modelos.

5.2.3 Uma Ferramenta que Automatizará as Traduções da linguagem RollGame na linguagem do SMV - StratAn-RollGame (Strategy Analyzed Using RollGame)

Uma ferramenta, denominada StratAn-RollGame (**Strat**egy **An**alyzed Using **RollGame**), foi desenvolvida visando automatizar as traduções de RollGame na linguagem de especificação de modelos do SMV, assim, como a execução do verificador de modelos. Para analisar um jogo é necessário apenas fornecer o jogo sem ou com estratégias, definido em RollGame, e as análises em CTL. Utilizou-se a técnica de transformação [9] [8] para gerar o código em SMV

a partir de RollGame. A ferramenta usada para a transformação foi a TXL (Tree eXchange Language) na versão 8.0[6].

A tradução de *GAL* para sentença de transições de estados ainda não foi implementada. Desta forma, a ferramenta só consegue analisar as fórmulas de *GAL* que já sejam sentenças de transições de estados.

Por que realizar as análises em CTL e não em *GAL*?

GAL poderia ser utilizada para analise de jogos sem estratégias sem nenhum problema, desde que a tradução de *GAL* em sentenças de transição de estados esteja implementada e a mesma tenha uma semântica. Contudo, o mesmo não ocorre para jogos com estratégias, visto que não foi implementado um verificador de modelos para *GAL*. Como utilizou-se o verificador de modelos SMV para realizar as análises é necessário utilizar a lógica CTL. Isto pode ser visto, por exemplo, que em CTL não há variáveis enquanto que em *GAL* elas podem ocorrer.

.

[6]TXL é uma linguagem funcional híbrida, para transformação fonte-a-fonte, baseada em regras e desenvolvida na Queen's University at Kingston, Canadá [15].