INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA

Verificação Formal de Programas com SPARK2014

PAULO MANUEL FERNANDES DE ASSUNÇÃO
Outubro de 2017

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Verificação Formal de Programas com SPARK2014

Paulo Manuel Fernandes de Assunção

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Prof. Doutor Jorge Coelho

Co-orientador: Doutor David Pereira

Júri:

Presidente:

[Nome do Presidente, Categoria, Escola]

Vogais:

[Nome do Vogal1, Categoria, Escola]

[Nome do Vogal2, Categoria, Escola] (até 4 vogais)

Porto, Outubro 2017

Dedicatória

À minha companheira, Mira, pelo seu apoio e incentivo.

Resumo

A verificação formal de software é uma área da Ciência de Computadores com grande importância no desenvolvimento de projetos de alta integridade. Ao adotar a verificação formal como uma das componentes dos projetos de software, aumenta-se a compreensão da natureza e propósito do programa por parte do programador e faz com que este se foque na precisão e consistência. Por outro lado, obriga também o programador a optar por soluções mais simples e que respeitem os requisitos de implementação, pois quanto mais complexo for o código desenvolvido maior será o esforço de prova e a dificuldade da mesma ser obtida de forma automática. Uma consequência é a minimização do risco de introdução de erros e maior facilidade na manutenção do programa.

A linguagem SPARK e o seu conjunto de ferramentas favorecem o desenvolvimento de programas da forma que é descrita no parágrafo anterior. Face às características da linguagem que impõe restrições, muitos erros de programação que são comuns noutras linguagens não acontecem.

O objetivo desta dissertação é o de estudar a usabilidade do SPARK2014 na implementação de estruturas de dados e algoritmos simples que são tipicamente utilizados no desenvolvimento de sistemas críticos ou de elevada integridade. Serão desenvolvidas especificações formais para os algoritmos em estudo, bem como implementações destes que são consistentes com essas mesmas especificações.

Por fim será feita uma análise a todo o processo, no sentido de obter indicadores do real valor do SPARK2014 no desenvolvimento de aplicações mais complexas.

Palavras-chave: Verificação formal, VCs, SPARK, Alta integridade, Why3

Abstract

Formal verification of software is an area of Computer Science with increasing importance and especially relevant for high-integrity projects. By adopting formal verification, programmers are brought to know better the nature of the algorithms they are implementing which need to be accurate and consistent. These properties are usually not simple to verify and as result of such complexity programmers tend to choose simpler solutions to minimize verification costs. Simple solutions have clear advantages in terms of error detection and maintenance.

The Spark language and its toolset have become a popular solution among high-integrity software production industry since it helps to produce correct software. The new Spark 2014 is a completely new version of the tool with a completely new approach to the verification process.

The aim of this work is to study the usability of the new SPARK 2014 in implementing data structures and algorithms commonly used in the development of high integrity systems and understand the real advantages of the trade-off imposed by a new, more complex approach. This goal is achieved by developing specification and implementing a set of selected algorithms using Spark 2014.

Keywords: Formal Verification, VCs, SPARK, High Integrity Systems, Why3

Agradecimentos

Ao Instituto Superior de Engenharia do Porto, ao seu corpo docente e não docente, por proporcionar a possibilidade, a todos aqueles que como eu se encontram já trabalhar, de adquirir novos conhecimentos e competências ao promover o ensino, também, em horário pós-laboral. Este é um exemplo que deveria ser seguido por todas as Escolas públicas.

A todos os Professores e Colegas que me acompanharam ao longo deste percurso de Mestrado, e que de uma forma ou de outra contribuíram para a minha formação, quer académica quer pessoal.

Por fim, um agradecimento muito especial aos meus orientadores, Prof. Jorge Coelho e Prof. David Pereira, pela sua orientação, incentivo e disponibilidade ao longo de todo o processo.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código

Acrónimos e Siglas

Lista de Acrónimos

SPADE	Southampton Program Analysis Development Environment
ISO	International Organization from Standardization
SPARK	SPADE Ada Ratiocinative Kernel

Lista de Siglas

PVL	Program Validation Ltd
CbC	Correctness by Construction
DbC	Design-by-contract
VC	Verification Condition
VCGen	Verification Condition Generator
SMT	Satisfiability Modulo Theories

1 Introdução

Hoje em dia os sistemas computacionais estão cada vez mais presentes na nossa vida. Na indústria, na área financeira, na saúde e nos serviços, são um componente no negócio. A forma como as empresas vêem o hardware e principalmente o software tem vindo a mudar, já começam a olhar para esta componente como um investimento, não como sendo um custo. A aquisição de tecnologias de informação TI têm vindo a aumentar de ano para ano [1][2].No entanto, o desenvolvimento de software tem custos elevados e as empresas pretendem que um produto adquirido não tenha falhas.

Existem áreas de negócio em que as falhas de software não são de todo admissíveis, como no caso da indústria aeroespacial, e da indústria aeronáutica, entre muitas outras. No desenvolvimento de software industrial para sistemas complexos os custos de testes e validação de resultados são elevados, sendo que normalmente uma grande fatia do orçamento é destinada a esta fase do desenvolvimento. O tempo e os testes necessários para garantir que todas as exigências foram cumpridas baseadas numa amostra representativa do contexto operacional, e que todos os aspetos do sistema foram cobertos, são consideráveis. Esta abordagem tem desvantagens, visto que a utilização de mecanismos de testes apenas aborda a identificação de erros existentes e não a sua ausência e também porque os erros são detetados numa fase adiantada do projeto ou já em produção. Caso a deteção seja na fase de projeto, esta pode implicar voltar atrás e refazer o projeto, caso já seja na fase produção/comercialização pode significar paragem e/ou inoperacionalidade dos equipamentos desenvolvidos [3] e grandes perdas para as empresas.

Na indústria aeronáutica, por exemplo, a procura de sistemas cada vez mais fiáveis, seguros e complexos aumenta o tempo e os custos de desenvolvimento. Por outro lado, existe a pressão, por parte da indústria, para que o tempo de desenvolvimento do produto diminua. Torna-se então necessário que o tempo de testes e validação seja encurtado, pois trata-se de uma componente com peso muito grande em termos de projeto [4]. Se forem utilizadas metodologias na construção de software que permitam a deteção de erros numa fase inicial, estas podem contribuir para a diminuição de tempo na componente de testes e validação.

É neste aspeto que a verificação formal de software é uma preciosa ajuda no desenvolvimento de projetos de software. Esta é uma metodologia que visa assegurar que o código está correto e de acordo com as especificações impostas. O uso desta metodologia permite diminuir o tempo e custos com testes e validação.

A verificação formal de um componente de software pode ser feita assim que este está escrito, mesmo antes de compilar, o que permite o alerta precoce de erros de programação. É possível aplicar esta metodologia a algumas partes do sistema, enquanto outros componentes ainda estão em desenvolvimento.

Esta metodologia é usada no desenvolvimento de sistemas de alta integridade, que normalmente têm custos de projeto elevados, pois tratam-se de sistemas em que uma falha pode causar perda de vidas humanas, danos ambientais, ou perdas financeiras e materiais irrecuperáveis. Nestes sistemas é vital que o software se comporte previsivelmente. Um sistema que provoca um erro em tempo de execução não se pode considerar previsível. Portanto é necessário eliminar todas as formas de comportamento imprevisível antes do software ser implementado.

O processo de verificação formal tem como função demonstrar, usando metodologias e abordagens bem fundadas em conceitos matemáticos. Estas metodologias assentam em noções como pré-condições que têm de ser satisfeitas antes da execução da execução do programa, e pós-condições que após a execução deste sempre se verificam. A exequibilidade da demonstração vai depender do nível de detalhe da especificação, que pode ir da visibilidade de variáveis até ao comportamento funcional.

A linguagem SPARK[5] foi desenvolvida com o objetivo de servir de ferramenta para a implementação de sistemas cuja integridade e segurança são fundamentais, e que permitisse a verificação formal do código. Apesar de ser um subconjunto da linguagem de programação Ada, o SPARK promove o conceito de CbC, com foco na prevenção do erro ao invés da sua deteção, o que obriga o programador a seguir um conjunto de restrições que eliminam muitas das fontes dos erros de programação que são comuns durante o processo de desenvolvimento de software. A linguagem SPARK tem sido usada pela indústria com sucesso em vários projetos e o seu conjunto de ferramentas para verificação e demonstração formal têm mostrado ser eficazes na análise de código.

Em 2014 foi lançado o SPARK2014 [6] que sendo ainda um subconjunto do Ada 2012, tem um maior âmbito. A nova versão do SPARK sofreu bastantes melhoramentos, dos quais se destacam a utilização de expressões de contratos formais (pré-condições, pós-condições, entre outras) na própria linguagem, resultado da maior convergência com a linguagem Ada, ao contrário do que ocorria no passado onde estes contratos eram especificados através de uma sub-linguagem de comentários com uma estrutura específica.

Outra grande alteração em relação às versões anteriores foi a opção pela utilização da plataforma de verificação Why3 [7] em detrimento das anteriores ferramentas de verificação.

Esta última escolha teve impacto profundo na forma como agora a verificação é conduzida no contexto de programas escritos em SPARK.

1.1 Objetivos

O objetivo desta dissertação é o de estudar a usabilidade do SPARK2014 na implementação de estruturas de dados e algoritmos que são utilizados no desenvolvimento de sistemas críticos ou de elevada integridade. Os algoritmos a implementar serão objeto de especificação formal. No desenvolvimento destes algoritmos serão usadas as ferramentas de verificação do SPARK2014 para garantir que a especificação e as várias implementações são, de facto, compatíveis.

A introdução de contratos formais na própria linguagem e a utilização da plataforma de verificação Why3 permitirá uma comparação com versões anteriores da linguagem de forma a avaliar o modo como é feita a verificação do código. Esta opção provocou alterações na forma como os contratos são escritos, pelo que deverá ser comparada a clareza da versão atual com anteriores.

A motivação para este trabalho vem da necessidade de entender as mudanças consideráveis que foram introduzidas no SPARK 2014 e que vantagens trazem para o domínio da programação de sistemas de elevada integridade. Esta é uma área da engenharia que, ainda não sendo largamente utilizada pelos profissionais da indústria, vem ganhando espaço face às cada vez maiores exigências de qualidade no software.

Por fim, analisar todo o processo levado a cabo durante a dissertação, no sentido de tirar conclusões que providenciem indicadores do real valor do SPARK2014 no desenvolvimento de aplicações mais complexas.

1.2 Estrutura

Este documento está organizado em seis capítulos.

No capítulo 1, “Introdução”, faz-se uma breve descrição do que é a verificação formal e quais os seus objetivos, da linguagem SPARK e dos objetivos a atingir nesta dissertação.

No capítulo 2, “Contexto”, contextualiza-se a verificação formal de software e a linguagem SPARK.

No capítulo 3, “SPARK”, é feita uma descrição da linguagem SPARK e das ferramentas usadas pela linguagem até à versão SPARK2005.

O capítulo 4, “SPARK2014 e Why3”, é dedicado à versão SPARK2014 e à plataforma Why3. São abordadas as alterações introduzidas na linguagem e o uso do Why3 como ferramenta de verificação.

No capítulo 5, “Verificação de estrutura de dados em SPARK2014”, são definidas as estruturas a verificar, a sua implementação e análise de resultados.

O capítulo 6, “Evolução do SPARK”, aborda as principais diferenças entre a versão 2005 e a 2014.

Por fim, no capítulo 7, “Conclusões”, são apresentadas as conclusões resultantes do trabalho realizado para esta dissertação, com a avaliação dos dados obtidos.

2 Contexto

A verificação formal de software é uma área da Ciência de Computadores que começou a ser estudada nos anos 60. Tem como objetivo a construção de algoritmos seguros e confiáveis. É baseada em formalismos matemáticos para a especificação, desenvolvimento e verificação de software.

Em 1969, Hoare propõe um conjunto regras, conhecidas como Lógica de Hoare [8], com o objetivo de garantir a correção do software. Esta ideia tem como base o trabalho desenvolvido por Floyd para fluxogramas[9].

O triplo de Hoare [8] é componente nuclear das formulações propostas e introduziu a notação

$$P\ \{Q\}\ R$$

e cuja interpretação é da correção parcial de programas e que deve ser interpretada da seguinte forma:

> "Se a asserção P é verdadeira antes do início do programa Q, então a asserção R será verdadeira sempre que programa terminar." (tradução nossa).

Esta noção é parcial pois não existe obrigatoriedade que o programa termine em todas as suas execuções. Existem, no entanto, extensões a este conceito em que a noção de terminação do programa é tida em consideração.

O uso de pré-condições, pós-condições e outras anotações, como inicialização de variáveis e/ou dependência destas entre si, no desenvolvimento do software, além de descrever o comportamento do programa pode ser visto também como um resumo/documentação rigorosa e não ambígua do que o algoritmo precisa, o que produz e o que altera. As anotações são também um elemento importante para utilização de ferramentas automáticas de verificação, que permitem avaliar se o comportamento do programa está de acordo com estas. Se as asserções são demonstráveis através das regras da Lógica de Hoare, então podemos

afirmar que o programa está correto de acordo com a sua especificação. O código presente na figura seguinte, em SPARK2014, é a soma de 2 inteiros e responde ao problema:

$$\forall\,(x, y \geq 0\ \wedge x, y \leq 1000) \wedge\ x = x + y\ \Longrightarrow x \leq 2000$$

```
Package Soma
with SPARK_Mode
is
    procedure Adiciona (x : in out Integer; y : in Integer)
       with
       pre => ((x >= 0 and x <= 1000) and (y >= 0 and y <= 1000)),
       post => (x = x'Old + y) and (x <= 2000);
end Soma;

Package body Soma
with SPARK_Mode
is
    Procedure Adiciona (x : in out integer; y : in Integer)
    is
    begin
       x := x + y;
    end Adiciona;
end Soma;


gnatprove -PD:soma.gpr --level=0 --ide-progress-bar -U --report=all
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
soma.adb:7:16: info: overflow check proved
soma.ads:7:17: info: postcondition proved
soma.ads:7:28: info: overflow check proved
Summary logged in D:\
```

Figura 1 – Resolução do problema Soma em SPARK

As anotações presentes na especificação, na forma de pré e pós-condições, definem os valores de entrada e descrevem qual o resultado esperado. Este código submetido à ferramenta de prova não apresenta qualquer erro.

2.1 Verificação formal de software

O desenvolvimento de software crítico ou de alta integridade para a indústria de defesa, aeronáutica, saúde, etc., requer que este seja certificado como sendo correto, seguro e fiável, e também que obedeça a critérios internacionalmente estabelecidos e descritos em standards que regulam as aplicações nestas áreas [10].

Os custos de desenvolvimento deste tipo de soluções são normalmente elevados, pelo que a deteção de erros numa fase inicial do projeto é essencial para que estes não cresçam desmesuradamente e se tornem virtualmente impossíveis de detetar.

Detetar erros em tempo de execução ou não detetá-los de todo, pode acarretar custos não apenas de desenvolvimento, mas de perdas materiais (caso do voo do Ariane5, entre outros) e de vidas, caso dos misseis Patriot na guerra do Golfo [11][12][13].

No desenvolvimento de um qualquer sistema computacional, critico ou não, se forem também usadas técnicas de verificação formal, construir-se-ão sistemas mais seguros, fiáveis e corretos. Linguagens como o SPARK e suas ferramentas permitem atingir estes objetivos, produzindo um software com mais qualidade, e menores custos de verificação.

2.2 Um pouco de história do SPARK

Nos finais dos anos 80, um grupo da Universidade de Southampton liderado por Bernard Carré, e mais tarde na PVL (foi adquirida pela Praxis Critical Systems em 1995, atualmente Altran UK), desenvolveu um sistema de verificação de programas baseada na Logica de Hoare para um subconjunto de Pascal. A este sistema foi dado o nome de SPADE[14], [15].

Tendo chegado à conclusão que a linguagem Pascal não era a mais adequada, foi necessário encontrar uma alternativa. A linguagem C não foi considerada pelo facto de ser muito permissiva e não ter um padrão definido internacionalmente. A opção recai sobre a linguagem Ada83, que em 1983 se torna um standard ANSI [16] e em 1987 passa a ser também padrão ISO [17]. Para além de ser uma linguagem já definida segundo os padrões ANSI e ISO, tinha um forte apoio da indústria da área do software critico e permitia o encapsulamento da informação devido à distinção entre especificação e implementação do código de programa. Outra característica interessante era o facto de ser uma linguagem com uma sintaxe fácil e passível de ser analisada, quer pelo programador/analista, quer através de um programa de computador.

Após um estudo profundo foi definido um subconjunto da linguagem de programação Ada. Neste subconjunto não foram consideradas características tais como, recursividade e alocação dinâmica de memória, além de impor certas restrições, como a dimensão dos arrays ter de ser sempre definida estaticamente, por exemplo. A verificação de propriedades de concorrência foi também posta de parte.

Além das restrições mencionadas, foram adicionadas anotações na forma de comentários Ada para permitir a especificação de fluxo de dados, com o propósito de criar uma linguagem robusta e inequívoca. Foram criadas as ferramentas Examiner, Simplifier e Proof Checker [14] para que a verificação formal de programas fosse possível e produzisse resultados fiáveis. Em simultâneo, todos os programas SPARK seriam programas válidos em Ada, produzindo os mesmos resultados independentemente do compilador usado e do equipamento a que se destinava. Desta forma não seria necessário desenvolver um novo compilador.

No entanto, a linguagem SPARK não é apenas um subconjunto de Ada com um conjunto de anotações fornecendo informação extra sobre o programa. Deve ser vista, isso sim, como uma linguagem dirigida ao desenvolvimento de software fiável, utilizando somente um conjunto de

características do Ada menos rico que a totalidade das construções da linguagem que lhe dá origem, mas é essa característica restritiva que lhe permite uma análise eficiente de programas.

As características da linguagem e as ferramentas utilizadas promovem uma metodologia de *Construção Correta de Programas* (CbC) [18][19] e a implementação do conceito de *Programação por Contrato* (DbC)[20].

2.2.1 Construção Correta de Programas (CbC)

Esta metodologia tem o intuito de detetar e corrigir erros de logica o mais cedo possível no ciclo de desenvolvimento, ou seja, construir programas corretos desde início. Basear a fiabilidade do programa somente em testes não é a forma ideal. Para além destes serem demorados e terem um custo financeiro elevado, nem sempre detetam todos os possíveis erros[21].

O SPARK acolhe bem esta metodologia de escrita de software, não é uma linguagem ambígua, tem ferramentas que permitem a verificação estática, a verificação da existência de dependências no código e a sua correção, utiliza anotações que são auxiliares importantes para as ferramentas de verificação e que permitem também uma maior clareza para análise humana.

O uso de SPARK, seguindo este conceito, no desenvolvimento de software crítico tem mostrado que, embora o tempo de desenvolvimento aparentemente seja maior, o custo total da solução é menor, incluindo testes e implementação [4].

2.2.2 Programação por contrato (DbC)

Programação por contrato é uma metodologia para construção de software, proposta por Meyer [20] aquando do desenvolvimento da linguagem Eiffel, e tem como ideia base o estabelecimento de um contrato entre duas entidades. O contrato define as obrigações entre o cliente (aquele a quem se destina o software) e o fornecedor (quem desenvolve o software) que ambos têm de satisfazer.

Um exemplo simples de contrato que estabelecemos no dia-a-dia, quando pretendemos enviar uma carta (cliente) temos a obrigação de, pelo menos, indicar o destinatário. Os correios (fornecedor) por sua vez têm a obrigação de a fazer chegar ao destino. Para o cliente não é importante como a carta chega ao destino.

Ao desenvolver software usando DbC, para cada procedimento é estabelecido um conjunto de regras, contrato, na forma de pré-condições a satisfazer pelo cliente, ao fornecedor são impostas pós-condições que este tem de garantir que são satisfeitas após a execução do código. Os detalhes de como o procedimento é implementado não são importantes para o cliente. O artigo [22] define o contrato como:

A palavra "contrato" é usada no sentido de que a especificação de cada procedimento estabelece um acordo entre um procedimento de chamada e o procedimento invocado: se P1

chama P2, P1 deve garantir que a pré-condição de P2 está satisfeita imediatamente antes da chamada; a especificação de P2 garante que a pós-condição será satisfeita quando o controle é devolvido a P1. Este princípio é cuidadosamente explorado na chamada abordagem de desenho-por-contrato para desenvolvimento de software. (tradução nossa)

Os métodos de verificação de formal têm vindo a ganhar espaço nos projetos de software, pois como já referimos, a indústria exige que o software entregue seja certificado como correto e seguro. Uma outra razão é a de que a verificação formal se enquadra na metodologia DbC. O uso de anotações como complemento à verificação baseada em logica, permite uma maior clareza na definição dos contratos, assim como automatizar a verificação, ainda que não na sua totalidade havendo por vezes necessidade de intervenção humana para verificar obrigações de prova mais difíceis. Os avanços registados nos processos de prova automática com o aparecimento de uma categoria de ferramentas, SMT [23], é mais uma razão da adoção desta metodologia. Os novos SMT verificam a satisfação de fórmulas de primeira ordem contendo operações de várias teorias, como booleanos, vetores de bits, várias classes de aritmética, matrizes e tipo recursivo. São utilizados para apoiar a verificação dedutiva de software, e têm um papel importante na verificação de programas [22].

A linguagem SPARK, pelas suas características, permite o desenvolvimento de software usando estas metodologias. Na versão de 2014 recorre à plataforma de verificação externa Why3, que para além das suas bibliotecas pode recorrer a um conjunto de demonstradores de teoremas externos para aumentar a automação da construção de demonstrações de correção de programas.

2.3 Alguns projetos desenvolvidos em SPARK

Existem muitos mais projetos em que o SPARK foi usado, mas estes são de alguma forma marcos na história da linguagem:

- **SHOLIS**[24][25] – Ship/Helicopter Operational Limits Instrumentation System, foi o primeiro projeto importante desenvolvido em SPARK. É um sistema projetado para a marinha do Reino Unido que auxilia a tripulação do navio na segurança das operações de helicóptero no convés de voo. A conceção, especificação e desenvolvimento foi efetuado pela Praxis Critical Systems.

- MULTOS CA[24] – Multi-Application Operating System, é um sistema operativo para smart-card que permite a existência de várias aplicações num único smart-card. A linguagem SPARK foi usada em 30% do projeto.

- LOCKHEED C130J[24][26] – O Lockheed C130J é um modelo do avião militar e civil mais conhecido como "Hércules". O núcleo do novo sistema de aviónica é o Mission Computer (MC), que efetua a maior parte das funções críticas no avião. O software do MC (aproximadamente 80%) é desenvolvido em SPARK.

- Lunar IceCube[27] – Em Julho de 2015 a AdaCore anunciou o início do projeto Lunar IceCube, que utilizará os conjuntos de ferramentas GNAT Pro e SPARK para desenvolver o software de alta confiabilidade necessário para missões espaciais. Lunar IceCube é um sucessor de um projeto anterior patrocinada pela NASA, também desenvolvido usando GNAT Pro e SPARK, que tem transmitido com sucesso dados de um CubeSat em órbita terrestre desde seu lançamento em novembro de 2013.

3 SPARK

Ao longo dos anos foram surgindo novas versões, desde o SPARK83, seguida de SPARK95 e SPARK2005. Todas estas versões têm como base o Ada 83, Ada 95 e o Ada 2005 respetivamente. A versão SPARK2014, baseada em Ada 2012, sofreu grandes alterações face às versões anteriores e que abordaremos mais à frente.

3.1 Estrutura

O SPARK enquanto linguagem de desenvolvimento está estruturado em três grandes áreas:

- Tipos de dados;
- Controlo de fluxo e de dados;
- Packages e visibilidade de variáveis.

3.1.1 Tipos

O SPARK oferece, primitivamente, vários tipos simples como inteiros (com e sem sinal), vírgula flutuante e enumerações. Podem ser declarados subtipos que restringem o intervalo de valores que uma variável pode assumir.

```
I, J, K : Integer range 1 .. 10;          → declaração ilegal em SPARK

subtype Index is Integer range 1 .. 10; → forma correta de atribuir
I, J, K: Index;                             intervalo de valores a variável
```

Código 1 - Definição de limites do subtipo, retirados de [14]

Um ponto importante é que todos os tipos e subtipos são declarados e as restrições são estáticas, ou seja, são conhecidas antes do programa ser compilado e executado. Possui

também tipos compostos como arrays e estruturas. Os arrays podem ser indexados por inteiros ou por enumeração.

```
-- registo 1
subtype Days is integer range 1 .. 31;
type Months is (Jan, Feb, Mar, Apr, May, June,
              July, Aug, Sept, Oct, Nov, Dec);
subtype Years is Integer 1 .. 4000;

type Date is record
       Day : Days;
       Month : Months;
       Year  : Years;
end record;


-- registo 2
type Object is tagged record
       x_coord, y_coord : Float;
end record;



-- Extensão do registo 2
type Circle is new Object with record
       radius : Float
end record;


-- array 1
subtype Index is Integer range 1 .. 10;
subtype Capital_Letter is Character range 'A' .. 'Z';
type Upper_Case_Array is array (Index) of Capital_Letter;
Upper_Case_Table: Upper_Case_Array;


-- array 2
type Tuple is array (Integer range <>) of Real;
subtype Tuple_Index is Integer 1 .. 3;
subtype Tuplo is Tuple(Tuple_Index);
```

Código 2 - Definição de registos e arrays, exemplos retirados de [14]

Os limites dos arrays são estáticos, embora possam ser definidos sem qualquer restrição mas, neste caso particular, tem de ser criado um subtipo que defina os limites pretendidos para o array em causa (ver Código 2, array 2).

O tipo registo é também determinado estaticamente (ver Código 2 registo 1), os campos que o compõem são do tipo ou subtipo definidos anteriormente com limites bem balizados. Embora seja permitida a extensão do tipo de registo, a regra para a criação de tipos registo mantém-se (ver Código 2 registo 2 e extensão). O tipo **Object** difere do tipo **Date** na sua definição pois, ao ser usada a palavra “tagged”, indica que este pode ser estendido, ou seja, podem ser criados

novos registos que herdam as suas propriedades, como é o caso do tipo **Circle**. No entanto existem restrições ao seu uso, só podem ser declarados na especificação de uma package e não podem coexistir na mesma package. Assim, os tipos **Object** e **Circle** têm de ser declarados em packages diferentes.

O SPARK não fornece suporte para memória dinâmica e apontadores, garantindo desta forma que não há nenhum acesso indevido à memória. O facto de usar um modelo de tipos simples facilita a prova de várias propriedades de programas escritos em SPARK.

3.1.2 Controlo de fluxo e de dados

As declarações do SPARK, no que respeita ao controlo de fluxo para instruções condicionais, são as instruções IF, LOOP e CASE. Declarações do tipo GOTO e LABEL são proibidos, e a recursividade também não é permitida. Não existe tratamento de exceções, sendo uma linguagem que visa o desenvolvimento de algoritmos para sistemas críticos, estas não podem ocorrer, tais como buffer overflow, divisão por zero, entre outras. Subprogramas em SPARK são procedimentos e funções. São usadas anotações para descrever o fluxo de dados dentro e fora de subprogramas.

```
package dissertacao
is
 procedure Troca (x, y : in out float);
 --# derives x from y &
 --#         y from x;
 --# post x = y~ and y = x;
end dissertacao;

package body dissertacao
is

 procedure Troca(x, y : in  out float)
 is
        t : float;
 begin
        t := x;
        x := y;
        y := t;
 end Troca;
end dissertacao;
```

Código 3 - Controlo de Fluxo

No código acima podemos ver as anotações que nos indicam que o valor de x depende do valor de y e por sua vez y depende de x.

3.1.3 Packages e visibilidade de variáveis

Packages e variáveis privadas são os mecanismos suportados pela linguagem para controlo da visibilidade do código escrito em SPARK. O overloading de subprogramas é proibida e desta forma garante que o nome do subprograma é único dentro da package. A visibilidade também é controlada por anotações que garantem que variáveis globais não são usadas de forma incorreta.

Um package permite definir as especificações de procedimentos, funções e tipos que estão logicamente relacionados entre si. A implementação deste mecanismo permite a evocação dos procedimentos por outras aplicações, sem dar a conhecer o funcionamento interno dos mesmos.

Um package por norma é constituído por duas partes: uma especificação e uma implementação. Pode acontecer não existir a parte de implementação por o package só definir tipos de dados e não procedimentos e funções. O código seguinte mostra uma estrutura usual de um package.

```
package dissertacao
is
 procedure Troca (x, y : in out float);
 --# derives x from y &
 --#         y from x;
 --# post x = y~ and y = x;
end dissertacao;

package body dissertacao
is

 procedure Troca(x, y : in  out float)
 is
        t : float;
 begin
        t := x;
        x := y;
        y := t;
 end Troca;
end dissertacao;
```

Código 4 - Definição típica de uma package

3.2 Ferramentas do SPARK2005

O conjunto de ferramentas composto pelo Examiner, Simplifier, Proof Checker e Proof Obligation Summarizer (POGS) é o que permite verificar se o código está de acordo com as especificações da linguagem e correto. Não tendo um compilador próprio, é usado um compilador de Ada para compilação do código, houve necessidade de criar uma ferramenta que verificasse as restrições da linguagem, o Examiner. Este é também um gerador de condições de verificação (VCGen).

O Simplifier é uma ferramenta que tenta provar de forma automática todas as condições de prova (VC) geradas pelo Examiner.

O Proof Checker é um assistente de prova que auxilia a descartar as VCs que o Simplifier não conseguir resolver.

O Proof Obligation Summarizer é uma ferramenta auxiliar que produz um relatório com as VCs que foram provadas, as que continuam por provar e as que se verificaram falsas.

3.2.1 Examiner

Verifica se o código está conforme as regras e cumpre as restrições da linguagem, se este é consistente com anotações. Estas embora não sejam uma ferramenta no sentido lato são uma componente essencial na função desempenhada pelo Examiner. Na tabela seguinte pode-se ver as anotações usadas pela linguagem.

Tabela 1 - Anotações

Anotação	Descrição
--# global	Declara uma variável global que pode ser acedida pelos procedimentos
--# derives	Define as dependências entre variáveis
--# own	Declara variáveis de estado dento da package
--# inherit	Herança, permite o acesso a procedimento e/ou variáveis de outras packages
--# main_program	Indica que é o programa principal
--# initializes	Indica que uma variável é inicializada antes da execução do programa
--# hide	Identifica código que não é examinado pelo Examiner
--# pre	Condição necessária à execução programa
--# pos	Condição que deve ser satisfeita após a execução do programa
--# assert	Anotação usada em ciclos para geração de VCs
--# return	Define o resultado de uma função

As anotações a preto dizem respeito ao fluxo de dados e informação, as restantes são usadas pelo Examiner para verificação de código e comportamento do mesmo. As anotações permitem ao Examiner gerar VCs que têm de ser provadas como corretas, para que se possa afirmar que o programa está correto em função das anotações.

O Examine funciona da mesma forma que um compilador, ou seja, tem de ter acesso a todo o código relacionado com o que está a ser analisado. A análise efetuada pela ferramenta tem como base a especificação das interfaces, pelo que é necessária uma ordem no escrutínio do código:

- Ao analisar um procedimento é necessário que a especificação da interface esteja disponível;
- Ao analisar a especificação de um package *A* que herda de outra *B*, é obrigatório que a

especificação de *B* esteja disponível;

- Ao analisar um package filho, o Examiner necessita de aceder ao seu package mãe.

Ao proceder à análise de um programa o Examiner produz mensagens relativas a erros de sintaxe e erros de fluxo.

Em Código 5 apresenta-se um código que tem um erro de logica, este package compilado em Ada não produz nenhum erro, ao ser submetido ao Examiner produz várias mensagens. A Figura 1 apresenta os dados da compilação e da análise.

```
package dissertacao
is
	procedure Troca (x, y : in out float);
	--# derives x from y &
	--#         y from x;
	--# post x = y~ and y = x~;
end dissertacao;

package body dissertacao
is

	procedure Troca(x, y : in  out float)
	is
		t : float;
	begin
		t := x;
		x := y;
		y := x;
	end Troca;
end dissertacao;
```

Código 5 - Package compilada em Ada e analisada pelo Examiner

```
D:\_DISSERTACAO\DEMO\src>gcc -c -gnat05 -gnato DISSERTACAO.ADB

D:\_DISSERTACAO\DEMO\src>
D:\_DISSERTACAO\DEMO\src>
D:\_DISSERTACAO\DEMO\src>
D:\_DISSERTACAO\DEMO\src>spark DISSERTACAO.ADs DISSERTACAO.ADB
            ******************************************************
                              Examiner GPL 2012
              Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
            ******************************************************


                         DATE : 10-JAN-2016 01:14:34.34


            Examining the specification of package dissertacao ...

            Generating listing file DISSERTACAO.lst ...

            Examining the body of package dissertacao ...

   8            t := x;
```

```
                ^
!!!        Flow Error        : 10: Ineffective statement.

  11    end Troca;

!!!        Flow Error        : 35: Importation of the initial value of variable x
           is ineffective.

!!!        Flow Error        : 33: The variable t is neither referenced nor
           exported.

!!!        Flow Error        : 50: y is not derived from the imported value(s) of x.

???        Flow Error        :601: y may be derived from the imported value(s) of y.

           Generating listing file DISSERTACAO.lst ...

           Generating report file ...

    5 errors or warnings, comprising:
         5 flow errors


-----------End of SPARK Examination--------------------------------

D:\_DISSERTACAO\DEMO\src>
```

Figura 2 - Resultado da compilação em Ada e mensagens do Examiner

Como se pode observar, na primeira linha da figura anterior, a compilação do código não produz qualquer erro. De fato não existe qualquer erro de sintaxe, pelo que, não é assinalado qualquer erro. O Examiner pelo seu lado produz 5 erros:

- Assinala que existe uma declaração que não produz qualquer efeito;
- A importação de x não produz qualquer efeito, pois o seu valor inicial é substituído pelo valor inicial de y;
- A variável t nunca é exportada;
- O valor de y não deriva do valor inicial de x como é referido nas anotações;
- O valor de y pode derivar do valor inicial de y, o que em termos lógicos não faz sentido.

```
 package dissertacao
is
 procedure Troca (x, y : in out float);
 --# derives x from y &
 --#         y from x;
 --# post x = y~ and y = x;
end dissertacao;

package body dissertacao
is

 procedure Troca(x, y : in  out float)
 is
        t : float;
 begin
        t := x;
        x := y;
        y := t;
 end Troca;
end dissertacao;
```

Código 6 - Código corrigido

Após a correção do código foi executado novamente o Examiner que não produziu qualquer erro, como se pode observar na figura seguinte.

```
D:\_DISSERTACAO\DEMO\src>spark DISSERTACAO.ADs DISSERTACAO.ADB
           ****************************************************
                          Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ****************************************************


                      DATE : 11-JAN-2016 21:05:23.34


           Examining the specification of package dissertacao ...

           Generating listing file DISSERTACAO.lst ...

           Examining the body of package dissertacao ...

+++        Flow analysis of subprogram Troca performed
           (information-flow mode): no errors found.

           Generating listing file DISSERTACAO.lst ...

           Generating report file ...

   No errors or warnings


-----------End of SPARK Examination--------------------------------

D:\_DISSERTACAO\DEMO\src>
```

Figura 3 - Resultados do Examiner após análise do corrigido

3.2.2 Simplifier

O Examiner para além de verificar a conformidade do código gera também *condições de verificação* (VC) na forma de um ficheiro com extensão *vcg* (ver Anexo 1). As VCs geradas são analisadas pela ferramenta SPADE Automatic Simplifier, ou de forma abreviada Simplifier, com o objetivo de provar que estas são verdadeiras. Caso se verifiquem VCs que não sejam provadas pelo Simplifier, estas terão de ser provadas por outros meios.

A ferramenta gera um ficheiro com extensão *siv* com as VCs verificadas, que pode ser usado pelo Proof Checker ou por uma equipa de verificação e revisão de código (ver Anexo 2).

3.2.3 Proof Checker

As VCs que não foram provadas como verdadeiras são analisadas pelo Proof Checker que usa o ficheiro siv para proceder à análise das VCs. O Proof Checker é um assistente de prova interativo que ajuda na construção de provas definindo regras que não estão previstas no Simplifier. O uso desta ferramenta requer algum conhecimento e perícia.

Por vezes são geradas VCs que nem o Proof Checker resolve, pelo que é necessária intervenção de uma equipa de verificação e revisão de código para provar estas VCs são verdadeiras.

4 SPARK2104 e Why3

A versão SPARK2014, daqui em diante referenciada somente como SPARK, é a mais recente e a que mais alterações sofreu. Esta versão baseia-se na versão 2012 do Ada que introduziu o conceito *aspects*, que permite formalizar contratos com anotações que fazem parte da própria linguagem[28]. Além desta alteração outra tem um grande impacto, a adoção da plataforma de verificação Why3 em detrimento do conjunto de ferramentas que as anteriores versões utilizavam.

4.1 SPARK2014

A nova versão de SPARK 2014 é mais flexível do que a versão anterior SPARK 2005. A linguagem pode ser aplicada em vários domínios, desde servidores, sistemas embebidos até sistemas de alta integridade e/ou criticas. Com a nova versão SPARK o subconjunto de Ada, que define esta linguagem, cresceu substancialmente. Com a adoção do conceito aspects do Ada os contratos deixam de ser escritos na forma de comentários, possibilitando um conjunto mais rico para os programadores desenvolverem o seu software.

A inclusão desta propriedade no SPARK continua a permitir a análise de programas incompleta, abstração, e análise de fluxo de informação e verificação formal de uma aplicação contra uma especificação. O uso dos aspects requer um compilador Ada, como por exemplo o compilador GNAT 2012.

A maior alteração verificada no SPARK2014 é adoção de uma plataforma, externa mais flexível, Why3, para geração de VCs e verificação formal, bem como de testes. Esta alteração vai de encontro à certificação de software exigida indústria aeronáutica e pelas autoridades reguladoras através da norma DO-178C[10], o suplemento DO-333[10] que já permite a verificação formal de software como uma metodologia de prova.

Nesta versão do SPARK foram introduzidas algumas características/funcionalidades interessantes, quer em termos de prova, quer em termos de desenvolvimento. Falamos de Expression Functions e Ghost Functions[29]:

- Expression Functions são funções cuja implementação é dada com uma simples expressão, não havendo necessidade de escrever o corpo desta.

```
function Increment (X : Integer) return Integer is (X + 1);
```

Código 7 – Exemplo de Expression Function

- Ghost Functions são funções criadas somente com o propósito de verificação formal, ou seja, na compilação do programa este código não é incluído. Por vezes existe a necessidade de recorrer a esta funcionalidade. Quando a especificação é complexa e difícil de escrever, pode haver necessidade de criar este tipo de funções dividindo a especificação em módulos mais pequenos e fáceis de escrever e gerir. Este tipo de função só pode ser utilizado em ciclos, pré e pós-condições.

O SPARK possui um conjunto de ferramentas para desenvolvimento constituído por um ambiente de desenvolvimento integrado, GPS, um compilador, um gerador de VCs que é também ferramenta de prova e uma ferramenta para testes.

4.1.1 GNAT Compiler

O compilador GNAT[29] executa as tarefas de um compilador típico:

- Verifica se o programa está conforme com as regras semânticas e de sintaxe do Ada
- Gera o código executável.

O guia do utilizador do SPARK aconselha que o primeiro passo no desenvolvimento de um programa é usar o compilador para garantir que o código é válido em Ada.

4.1.2 GNATprove

GNATprove é a ferramenta de verificação para SPARK [29]. Pode ser executado em três modos diferentes:

- Verificação – verifica se um programa contém apenas o subconjunto de Ada definido para SPARK.
- Fluxo – Realiza a análise de código. Verifica a inicialização de variáveis, variáveis não usadas, as dependências de dados dos subprogramas.

- Prova – executa uma verificação formal do código utilizando o Why3 para a geração de VCs. É também com o Why3 que feita a verificação formal, esta ferramenta atua como intermediário entre o GNATProve e os SMTs disponíveis e/ou selecionados. A verificação formal permite avaliar se existe código que possa gerar um erro de tempo de execução, como divisão por zero, atribuição a uma variável de um valor que está fora do intervalo definido ou do seu tipo, indexação incorreta de arrays, ou overflow de uma expressão aritmética. Se o código contém declarações de propriedades funcionais, elas são verificadas.

4.1.3 GNATtest

Com o Ada 2012 surge o GNATtest, uma ferramenta para criação de unidades de teste de código Ada e por isso pode ser usado pelo SPARK. Nem sempre é possível verificar formalmente todo o código de um programa, há várias razões para que não seja possível fazer a prova para a resolução de um determinado problema, ou poderá ser mais económico testar o código do que verifica-lo formalmente. O GNATtest é a ferramenta usada para testar os programas que não são passíveis de verificação formal [29]. O GNATtest é uma ferramenta baseada na framework AUnit [30], [31] que gera testes unitários para o código desenvolvido.

4.2 Why3

O Why3 é uma plataforma para verificação dedutiva de programas [7], que sucede ao Why [32] [7]. Foram introduzidos novos recursos (numerosas extensões para a linguagem de entrada, uma nova arquitetura para ter acesso a ferramentas de prova externas, automatizadas e/ou interativas, e uma API bem concebida permitindo usar o Why3 como uma biblioteca de software. Uma característica importante é a modularidade, permitindo ao utilizador a possibilidade de reutilizar facilmente formalizações Why3 ou adicionar novas ferramentas de prova.

É uma ferramenta que produz VCs e tem a sua própria linguagem interna. Outras linguagens existentes podem ser compilada para a linguagem interna do Why3, o WhyML, que se trata de uma pequena linguagem de programação semelhante ao ML, com características imperativas, exceções e anotações. Possui ainda uma biblioteca padrão de teorias lógicas (inteiros e aritmética real, booleanos, conjuntos, etc.). Tal como o ML, junta expressões, instruções, variáveis locais e funções numa única classe, o que simplifica a manipulação de símbolos e limita o número de casos a considerar quando se calculam as condições de verificação. É usada como linguagem intermedia para verificar programas desenvolvidos em C, Java e Ada.

O Why3 não é em si uma ferramenta de prova, funciona como um intermediário entre as linguagens de desenvolvimento e demonstradores de teoremas desenvolvidos por terceiros.

Esta abordagem permite uma maior automatização de prova de VCs, tirando partido das diferentes capacidades de automação de demonstração dos demonstradores suportados.

Ferramentas de prova automática suportadas pela plataforma:

- Alt-Ergo, https://alt-ergo.ocamlpro.com/
- Beagle, https://bitbucket.org/peba123/beagle
- CVC3, http://www.cs.nyu.edu/acsys/cvc3/
- CVC4, http://cvc4.cs.nyu.edu/web/
- E-prover, http://www4.informatik.tu-muenchen.de/~schulz/E/E.html
- Gappa, http://gappa.gforge.inria.fr/
- Metis, http://www.gilith.com/software/metis/
- Metitarski, http://www.cl.cam.ac.uk/~lp15/papers/Arith/
- Princess, http://www.philipp.ruemmer.org/princess.shtml
- Psyche, http://www.lix.polytechnique.fr/~lengrand/Psyche/
- Simplify, http://kindsoftware.com/products/opensource/Simplify/
- SPASS, http://www.spass-prover.org/
- Vampire, http://www.vprover.org/
- veriT, http://www.verit-solver.org/
- Yices, http://yices.csl.sri.com/
- Z3, https://github.com/Z3Prover/z3

Ferramentas de prova interativa ou *Proof Assistants*:

- Coq, https://coq.inria.fr/
- PVS, http://pvs.csl.sri.com/
- Isabelle/HOL, http://isabelle.in.tum.de/

5 Verificação de estruturas de dados em SPARK2014

Neste capítulo serão apresentadas estruturas de dados e/ou algoritmos desenvolvidos em SPARK2014. Sempre que seja útil para uma melhor explicação, será apresentada também uma versão em SPARK2005 ou outra linguagem.

- As estruturas de dados e/ou algoritmos a desenvolver são:
- Pilha (Stack);
- Lista Circular (Circular Buffer);
- Fatorial
- N-Queens
- Produtor/Consumidor
- Stack Concorrente

É importante que os exemplos desenvolvidos possam ser replicados, assim, no Anexo VII apresenta-se todo o código fonte, as configurações dos projetos usadas no editor GPS e imagens dos ecrãs com a execução dos programas.

5.1 Pilha

A estrutura de dados Pilha - Stack é o termo comumente utilizado - é uma estrutura simples baseada no princípio de o último elemento colocado ser sempre o primeiro a ser retirado, e implementada e testada em várias linguagens.

A implementação da Stack normalmente é feita recorrendo a duas operações:

- Push que adiciona um elemento à pilha;
- Pop que retira o último elemento adicionado.

O código que se segue, em Stack2005 e Stack2014, acrescenta uma operação **Clear** (limpeza da pilha), que não interfere com o princípio de funcionamento da estrutura. Alem destes três procedimentos existem mais três funções:

- Top – retorna o último elemento da pilha;
- Full – verifica se pilha está cheia;
- Empty – verifica se pilha está vazia.

A diferença deste código relativamente a outras linguagens é a existência de contratos, que permitem às ferramentas do SPARK provar que o código está correto. Dito de outra forma, os resultados pretendidos são aqueles que efetivamente se obtêm.

Para se fazer prova da correção do código, ao nível do fluxo da informação, existem um conjunto de anotações que permitem que a ferramenta de verificação, GNATprove, analise o código e este se encontra de acordo com as mesmas. Nas versões anteriores de SPARK as anotações/contratos eram escritos sob a forma de comentários ADA que eram tratados pelo Examiner. Na versão 2014 é utilizada a nova funcionalidade do Ada 2012, a especificação **Aspects**, para elaborara as anotações/contratos.

No caso desta estrutura de dados, a diferença entre o SPARK2005 e o SPARK2014, ao nível da especificação, são somente a forma como são feitas as anotações/contratos, como se pode observar no código seguinte.

```
001     --SPARK 2005
002     package Stack2005
003     is
004         type Stack is private;
005
006         function Top (S : in Stack) return Integer;
007         function Empty (S : in Stack) return Boolean;
008         function Full (S : in Stack) return Boolean;
009
010         procedure Push (S : in out Stack; I : in Integer);
011         --# pre not Full(S);
012         --# post not Empty(S);
013         procedure Pop (S : in out Stack; I : out Integer);
014         --# pre not Empty(S);
015         --# post not Full(S);
016         procedure Clear (S : in out Stack);
017         --# post Empty(S);
018
019     private
```

```
020         Stack_Size : constant := 10;
021         type Pointer_Stack is range 0 .. Stack_Size;
022         subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
023         type Vector is array (Index_Stack) of Integer;
024
025         type Stack is record
026         Size_of_Stack : Pointer_Stack;
027         Data_in_Stack : Vector;
028         end record;
029
030     end Stack2005;
```

Código 8 – Stack2005.ads, especificação da pilha

```
001     -- SPARK 2014
002     package Stack2014
003     with SPARK_Mode
005     is
006         type Stack is private;
007
008         function Top (S : in Stack) return Integer;
009         function Empty (S : in Stack) return Boolean;
010         function Full (S : in Stack) return Boolean;
011
012         procedure Push (S : in out Stack; I : in Integer)
013         with
014             Pre => (not Full(S)),
015             Post => (not Empty(S));
016         procedure Pop (S : in out Stack; I : out Integer)
017         with
018             Pre => (not Empty(S)),
019             Post => (not Full(S));
020         procedure Clear (S : in out Stack)
021         with
022             Post => (Empty(S));
023     private
025         Stack_Size : constant := 10; -- tamanho maximo da stack
026         type Pointer_Stack is range 0 .. Stack_Size;
027         subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
030         type Vector is array (Index_Stack) of Integer;
028
029         type Stack is record
030         Size_of_Stack : Pointer_Stack := 0;
031         Data_in_Stack : Vector := Vector'(Index_Stack => 0);
032         end record;
033     end Stack2014;
```

Código 9 – Stack2014.ads, especificação da pilha

A tabela seguinte resume e mapeia as anotações de SPARK2005 para SPARK2014 existentes neste bloco de código.

Tabela 2 – Anotações na especificação de Stack2005/Statck2014

SPARK2005	**SPARK2014**
`--# pre`	`Pre =>`
`--# post`	`Post =>`

Neste código só são utilizadas pré-condições e pós-condições, que estão presentes nos ficheiros de especificação, Stack2005.ads e Stack2014.ads.

No que diz respeito à implementação, percebem-se algumas diferenças, quer nas anotações quer no código.

```
001     --SPARK 2005
002     package body Stack2005
003     is
004         procedure Push (S : in out Stack; I : Integer)
005         is
006         begin
007             S.Size_of_stack := S.Size_of_stack + 1;
008             S.Data_in_stack (S.Size_of_stack) := I;
009         end Push;
010
011         procedure Pop (S : in out Stack; I : out Integer)
012         is
013         begin
014             I := S.Data_in_stack (S.Size_of_stack);
015             S.Size_of_stack := S.Size_of_stack - 1;
016         end Pop;
017
018         procedure Clear (S : in out Stack)
019         is
020         begin
021             S.Size_of_Stack := 0;
022         end Clear;
023
024         function Top (S : in Stack) return Integer
025         --# return S.Data_in_stack (S.Size_of_stack);
026         is
027             v_top : integer := 0;
028         begin
029             if S.Size_of_stack > 0 then
030                 v_top := S.Data_in_stack (S.Size_of_stack);
031             end if;
032             return v_top;
033         end Top;
034
035         function Empty (S : in Stack) return Boolean
036         --# return S.Size_of_Stack = 0;
037         is
038         begin
039             return S.Size_of_Stack = 0;
040         end Empty;
041
042         function Full (S : in Stack) return Boolean
043         --# return S.Size_of_Stack = Stack_Size;
044         is
045         begin
046             return S.Size_of_Stack = Stack_Size;
047         end Full;
048
049     end Stack2005;
```

Código 10– Stack2005.ads, implementação da pilha

```
001     -- SPARK 2014
002     package body Stack2014
003     with SPARK_Mode
```

```
004     is
005         procedure Push (S : in out Stack; I : Integer)
006         is
007         begin
008             S.Size_of_stack := S.Size_of_stack + 1;
009             S.Data_in_stack (S.Size_of_stack) := I;
010         end Push;
011
012         procedure Pop (S : in out Stack; I : out Integer)
013         is
014         begin
015             I := S.Data_in_stack (S.Size_of_stack);
016             S.Size_of_stack := S.Size_of_stack - 1;
017         end Pop;
018
019         procedure Clear (S : in out Stack)
020         is
021         begin
022             S.Size_of_Stack := 0;
023         end Clear;
024
025         function Top (S : in Stack) return Integer
026         is
027         begin
028             if S.Size_of_stack > 0 then
029                 return S.Data_in_stack (S.Size_of_stack);
030             end if;
031             return 0;
032         end Top;
033
034         function Empty (S : in Stack) return Boolean is (S.Size_of_Stack = 0);
035
036            function Full (S : in Stack) return Boolean is (S.Size_of_Stack =
Stack_Size);
037
038     end Stack2014;
```

Código 11 – Stack2014.ads, implementação da pilha

Se analisarmos os dois blocos de código, de implementação, verificamos que na versão 2005 existem anotações ao contrário do que acontece na versão 2014.

As funções existentes na implementação da Pilha têm como objetivo provar que a implementação dos procedimentos está correta. Não tendo sido definidas anotações na especificação, na versão 2005, existe a necessidade de definir pós-condições na implementação, na forma de comentário **--# return**, de modo que, a função seja dada como provada e correta e possa ser usada como condição de verificação dos procedimentos. Na versão 2014, não existe a necessidade de definir pós-condições no corpo da função se esta for definida como Expression Function. Esta função é implementada com uma simples expressão, o seu resultado pode ser usado como pós-condição de um procedimento. Exemplo:

```
function Empty (S : in Stack) return Boolean is (S.Size_of_Stack = 0);
```

O resultado da função é Empty'Result = E, (E = (S.Size_of_Stack = 0)), que resulta da analise feita automaticamente pelas ferramentas de verificação do SPARK2014. Esta funcionalidade foi introduzida na versão 2014, o que torna mais simples a construção de código assim como a sua verificação.

Na função Top, na versão 2005, é declarada uma variável v_top para retornar o valor resultante da função. Isto acontece porque ao SPARK2005 só é permitido o uso uma vez da palavra reservada return por função e esta tem de ser a ultima frase. Em SPARK20014 esta restrição não existe.

A utilização de contratos garante que o procedimento está correto, a Figura 4 mostra o resultado obtido com a utilização da ferramenta GNATprove. A primeira chamada do GNATprove destinou-se a verificar se todas as variáveis foram inicializadas e são usadas corretamente no código. A segunda chamada da ferramenta teve como objetivo fazer prova que o programa está correto, o que se verificou. A verificação e prova do código em SPARK2005 consta dos Anexo III e IV.

```
gnatprove   -PD:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2014   Red\stack2014.gpr   --
mode=flow --ide-progress-bar
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: analysis of data and information flow ...
stack2014.ads:17:38: info: initialization of "I" proved
Summary logged in D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2014 Red\obj\
gnatprove\gnatprove.outgnatprove

PD:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2014  Red\stack2014.gpr  --ide-progress-bar
--report=all --level=2
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
stack2014.adb:8:44: info: range check proved
stack2014.adb:9:27: info: index check proved
stack2014.adb:15:32: info: index check proved
stack2014.adb:16:44: info: range check proved
stack2014.adb:29:38: info: index check proved
stack2014.ads:15:21: info: postcondition proved
stack2014.ads:17:38: info: initialization of "I" proved
stack2014.ads:20:21: info: postcondition proved
stack2014.ads:24:21: info: postcondition proved
Summary        logged        in        D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2014
Red\obj\gnatprove\gnatprove.out
```

Figura 4 – Resultado da verificação e prova do código usando GNATprove

5.2 Lista Circular

Uma Lista Circular(Circular Buffer), é uma estrutura de dados simples do tipo array. Os dados são lidos e escritos de forma circular, ou seja, ao atingir o fim do array numa operação de escrita ou leitura retorna ao início deste. O código seguinte implementa esta estrutura de dados em SPARK2014.

```
001     package ListaCircular2014
002     with SPARK_Mode
003     is
004
005         type ListaCircular is private;
006
007         procedure Guardar (CB : in out ListaCircular; Inteiro : in Integer)
008         with
009             Pre => (not ListaCheia (CB));
010
011         procedure Ler (CB : in out ListaCircular; Inteiro : out Integer)
012         with
013             pre => ( not ListaVazia (CB));
014
015         function ListaCheia (CB : in ListaCircular) return Boolean;
016         function ListaVazia (CB : in ListaCircular) return boolean;
017
018     private
019         ListaCirc_Size : constant := 10;
020         type Cauda_ListaCirc is range 0 .. ListaCirc_Size;
021         subtype Index_Cauda_ListaCirc
022             is Cauda_ListaCirc range 1 .. ListaCirc_Size;
023         type Vector is array (Index_Cauda_ListaCirc) of Integer;
024
025         type ListaCircular is record
026         Cabeca : Cauda_ListaCirc := 0;
027         Cauda  : Cauda_ListaCirc := 0;
028         Dados  : Vector := Vector'(Index_Cauda_ListaCirc => 0);
029         end record;
030     end ListaCircular2014;
```

Código 12 - ListaCircular.ads

```
001     package body ListaCircular2014
002     with SPARK_Mode
003     is
004
005         procedure Guardar (CB : in out ListaCircular; Inteiro : in Integer)
006         is
007         begin
008         if not ListaCheia(CB) then
009             CB.Cauda := CB.Cauda + 1;
010             CB.Dados (CB.Cauda) := Inteiro;
011             if CB.Cauda = CB.Dados'Length then
012             CB.Cauda := 0;
013             end if;
014         end if;
015         end Guardar;
016
017         procedure Ler (CB : in out ListaCircular; Inteiro : out Integer) is
018         begin
019         Inteiro := 0;
020         if CB.Cabeca /= CB.Cauda then
021             CB.Cabeca := CB.Cabeca + 1;
022             Inteiro := CB.Dados (CB.Cabeca);
023             if CB.Cabeca = CB.Dados'Length then
024             CB.Cabeca := 0;
025             end if;
026         end if;
027         end Ler;
028
029         function ListaCheia (CB : in ListaCircular) return Boolean is
030         cheia : Boolean := False;
031         begin
032         if CB.Cauda + 1 = CB.Cabeca then
033             cheia := true;
```

```
034         elsif CB.Cauda = CB.Dados'Length - 1 and CB.Cabeca = 0 then
035             cheia := true;
036         end if;
037         return cheia;
038         end ListaCheia;
039
040         function ListaVazia (CB : in ListaCircular) return boolean is
041         vazia : Boolean := False;
042         begin
043         if CB.Cabeca = 0 and CB.Cauda = 0 then
044             vazia := true;
045         end if;
046         return vazia;
047         end ListaVazia;
048     end ListaCircular2014;
```

Código 13 – ListaCircular.adb

Como se pode observar na Figura 5, a ferramenta GNATprove não considera este código como correto. As anotações estabelecidas não são suficientes para que a ferramenta de prova possa considerar o código formalmente correto. Como se pode observar pelos erros detetados, linhas 9 e 21, o GNATprove não garante que o resultado da soma não ultrapasse o valor máximo permitido, podendo gerar um valor para índice do array inválido.

```
gnatprove                                                                    -
PD:\_DISSERTACAO\_EXEMPLOS_TESE\ListaCircular\ListaCircular2014\ListaCircular2014.gpr  --ide-
progress-bar --level=2 --report=all
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
listacircular2014.adb:9:34: medium: range check might fail
listacircular2014.adb:10:25: info: index check proved
listacircular2014.adb:21:36: medium: range check might fail
listacircular2014.adb:22:36: info: index check proved
listacircular2014.adb:37:16: info: initialization of "cheia" proved
listacircular2014.adb:46:16: info: initialization of "vazia" proved
listacircular2014.ads:11:47: info: initialization of "Inteiro" proved
Summary                                  logged                                  in
D:\_DISSERTACAO\_EXEMPLOS_TESE\ListaCircular\ListaCircular2014\obj\gnatprove\gnatprove.out
[2016-01-16 16:29:06] process terminated successfully, elapsed time: 01.58s
```

Figura 5 – Resultado da verificação de ListaCircular

O código foi revisto, tendo sido alterada a implementação da Lista Circular. Como se pode observar em Código 14, foram feitas algumas alterações ao código e introduzidas condições (aspects) de forma a obter uma verificação sem erros.

```
001     package body ListaCircular2014
002     with SPARK_Mode
003     is
004
005         procedure Guardar (CB : in out ListaCircular; Inteiro : in Integer)
006         is
007         begin
008         if CB.Cauda < ListaCirc_Size then
009             pragma assert (not ListaCheia (CB));
```

```
010             CB.Cauda := CB.Cauda + 1;
011               pragma assert (if CB.Cauda < ListaCirc_Size then CB.Cauda + 1 <=
ListaCirc_Size );
012             CB.Dados (CB.Cauda) := Inteiro;
013             if CB.Cauda = CB.Dados'Length then
014             CB.Cauda := 0;
015             end if;
016         end if;
017         end Guardar;
018
019
020
021         procedure Ler (CB : in out ListaCircular; Inteiro : out Integer) is
022         begin
023         Inteiro := 0;
024     --  if CB.Cabeca /= CB.Cauda then
025         if CB.Cabeca < ListaCirc_Size then
026             pragma assert (not ListaVazia (CB));
027             CB.Cabeca := CB.Cabeca + 1;
028              pragma assert(if CB.Cabeca < ListaCirc_Size then CB.Cabeca + 1 <=
ListaCirc_Size);
029             Inteiro := CB.Dados (CB.Cabeca);
030             if CB.Cabeca = CB.Dados'Length then
031             CB.Cabeca := 0;
032             end if;
033         end if;
034         end Ler;
035
036
037
038         function ListaCheia (CB : in ListaCircular) return Boolean is
039         cheia : Boolean := False;
040         begin
041         if CB.Cauda + 1 = CB.Cabeca then
042             cheia := true;
043         elsif CB.Cauda = CB.Dados'Length - 1 and CB.Cabeca = 0 then
044             cheia := true;
045         end if;
046         return cheia;
047         end ListaCheia;
048
049
050         function ListaVazia (CB : in ListaCircular) return boolean is
051         vazia : Boolean := False;
052         begin
053         if CB.Cabeca = 0 and CB.Cauda = 0 then
054             vazia := true;
055         end if;
056         return vazia;
057         end ListaVazia;
058
059     end ListaCircular2014;
```

Código 14 – Nova implementação de ListaCircular.adb

Foram alteradas as condições de teste, linhas 8 e 25, e acrescentadas as anotações pragma assert, anotações check em SPARK2005. O pragma assert gera uma nova condição de verificação para provar a expressão que o antecede. Para definir estes contratos utilizaram-se

as funções ListaCheia e ListaVazia, o que permitiu ao GNATprove validar as expressões condicionais precedentes.

Para o GNATprove poder verificar que CB.Cauda e CB.Cabeca não ultrapassam o valor máximo permitido, foi adicionado um pragma assert com condições lógicas que geram novas VC's que permitem validar e provar como certa as operações de soma, linhas 10 e 27. O resultado da verificação e prova consta da figura seguinte.

```
gnatprove                                                                    -
PD:\_DISSERTACAO\_EXEMPLOS_TESE\ListaCircular\ListaCircular2014\ListaCircular2014.g
pr --ide-progress-bar --level=2 --report=all
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
listacircular2014.adb:9:13: info: assertion proved
listacircular2014.adb:10:34: info: range check proved
listacircular2014.adb:11:13: info: assertion proved
listacircular2014.adb:12:25: info: index check proved
listacircular2014.adb:26:13: info: assertion proved
listacircular2014.adb:27:36: info: range check proved
listacircular2014.adb:28:13: info: assertion proved
listacircular2014.adb:29:36: info: index check proved
listacircular2014.adb:46:16: info: initialization of "cheia" proved
listacircular2014.adb:56:16: info: initialization of "vazia" proved
listacircular2014.ads:11:47: info: initialization of "Inteiro" proved
Summary                                  logged                                  in
D:\_DISSERTACAO\_EXEMPLOS_TESE\ListaCircular\ListaCircular2014\obj\gnatprove\gnatpr
ove.out
[2016-03-17 00:58:18] process terminated successfully, elapsed time: 01.72s
```

Figura 6 – Resultado da verificação após alterações a ListaCircular

Tabela 3 – Anotações na implementação de ListaCircular e o equivalente em SPARK2005

SPARK2005	SPARK2014
--# check	pragma assert

5.3 Fatorial

O Fatorial de um número n é o produto de todos os inteiros positivos menores ou iguais a n. O algoritmo que implementa este produto é sobejamente conhecido e muito simples. A implementação pode ser feita utilizando um ciclo ou através da recursividade.

```
001     package Factorial2005 is
002         --# function Fact(N : Natural) return Natural;
003         function Factorial (N : Natural) return Natural;
004         --# pre (N >= 0 and N <= 12);
005         --# return Fact(N);
006
007     end Factorial2005;
```

Código 15 – Factorial20015.ads

```
001     package Factorial2014
002     with SPARK_Mode
003     is
004
005         function Factorial (N : in Natural) return Natural
006         with
007         Depends => (Factorial'Result => N),
008         pre => (N >= 0 and N <= 12),
009          post => (Factorial'Result >= 0 and Factorial'Result - Natural'Last <=
0);
010
011     end Factorial2014;
```

Código 16 – Factorial2014.ads

Ao nível da especificação, as diferenças a versão 2005 e a 2014 prendem-se com as anotações. Na versão 2005 são utilizadas três tipos de anotações, uma função de prova, uma pré-condição, uma anotação de retorno que funciona como pós-condição e usa a função de prova para validar a correção do algoritmo. A versão 2014 não utiliza uma função de prova, mas acrescenta a anotação Depends que indica que o resultado da função fatorial depende do valor de entrada (N).

Tabela 4 – Anotações na especificação de Factorial/Factorial2014

SPARK2005	**SPARK2014**
--# function	With Ghost
--# pre	Pre =>
--# return	Atributo 'Result

Em SPARK2014 não existem funções de prova, e embora as funções Ghost não tenham uma equivalência direta podem ser usadas como tal. Na versão 2014 não foi utilizada uma função Ghost, a prova foi efetuada recorrendo a outras anotações na implementação. O atributo 'Result é usado nas pós-condições na versão 2014 e pode-se dizer que é o equivalente à anotação --#return em 2005, que funciona como uma pós-condição. A utilização destas anotações destina-se a provar que a implementação produz o resultado previsto.

A implementação é diferente nas duas versões, quer no código quer nas anotações como se pode observar em Código 17 e Código 18. A versão 2014 ao não usar uma função de prova recorre a mais anotações para validação do código.

```
001     package body Factorial2005 is
002         --# function Fact(N : Natural) return Natural;
003         function Factorial (N : Natural) return Natural
004         is
005             Result : Natural := 1;
006         begin
007             for I in Natural range 1 .. N loop
008                 exit when I > N;
009                     Result := Result * I;
010                     --#assert I > 0 and Result = Fact(I);
011             end loop;
```

```
012             return Result;
013         end Factorial;
014
015     end Factorial2005;
```

Código 17 – Factorial2005.adb, implementação

```
001     package body Factorial2014
002     with SPARK_Mode
003     is
004         function Factorial (N : in Natural) return Natural
005         is
006             r     : Natural := 1;
007             r_old : Natural := r with Ghost;
008             i     : Natural := 0;
009
010         begin
011         loop
012             pragma Loop_Invariant (I >= 0 and  I <= N);
013             exit when i = N;
014                 i := i + 1;
015                 r_old := r;
016                 r := r * i;
017
018              pragma Assert (if i <= N and r_old / N <= Natural'Last and r_old *
i <= Natural'Last then r <= Natural'Last);
019            pragma Annotate (GNATprove, Intentional, "", "r always < Natural'Last
because N<= 12");
020         end loop;
021         return r;
022         end Factorial;
023
024     end Factorial2014;
```

Código 18 – Factorial2014.adb, implementação

Na versão 2005 temos somente duas anotações, --# function e --#assert. A função de prova declarada tanto na especificação como na implementação destina-se somente a ser usada pelo Examiner. SPARK2005 não permite recursividade, no entanto é permitido nas anotações, para uso pelas ferramentas de verificação e prova. A anotação --#assert indica que a cada iteração o valor de I é o Resultado = Fact (I). Os resultados da verificação e prova constam dos anexos V e VI.

- Na implementação em SPARk2014 pode observar-se a utilização de três anotações pragma:
- pragma Loop_Invariant, permite estabelecer uma asserção que tem necessariamente de ser verdadeira na entrada e à saída de um ciclo.
- pragma Assert, esta anotação gera uma VC para provar que a expressão SPARK que a precede está correta.
- pragma Annotate, esta anotação permite ultrapassar mensagens de verificação geradas

pelo GNATprove. A figura seguinte é o resultado da verificação do programa Factorial2014 com a linha 19 comentada.

```
Gnatprove                                                                    -
PD:\_DISSERTACAO\_EXEMPLOS_TESE\Factorial\Factorial2014\factorial2014.gpr
--ide-progress-bar --report=all --level=2
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
factorial2014.adb:7:28: info: initialization of "r" proved
factorial2014.adb:12:13: info: loop invariant preservation proved
factorial2014.adb:12:13: info: loop invariant initialization proved
factorial2014.adb:12:36: info: initialization of "i" proved
factorial2014.adb:13:23: info: initialization of "i" proved
factorial2014.adb:14:18: info: initialization of "i" proved
factorial2014.adb:14:20: info: overflow check proved
factorial2014.adb:15:22: info: initialization of "r" proved
factorial2014.adb:16:18: info: initialization of "r" proved
factorial2014.adb:16:20: medium: overflow check might fail
factorial2014.adb:16:20: info: range check proved
factorial2014.adb:16:22: info: initialization of "i" proved
factorial2014.adb:18:13: info: assertion proved
factorial2014.adb:18:31: info: initialization of "i" proved
factorial2014.adb:18:42: info: initialization of "r_old" proved
factorial2014.adb:18:48: info: division check proved
factorial2014.adb:18:78: info: overflow check proved
factorial2014.adb:21:16: info: initialization of "r" proved
factorial2014.ads:5:14: info: "Factorial2014" requires body ("Factorial" requires
completion)
factorial2014.ads:9:17: info: postcondition proved
Summary                                  logged                                 in
D:\_DISSERTACAO\_EXEMPLOS_TESE\Factorial\Factorial2014\obj\gnatprove\gnatprove.ou
t
[2016-12-16 23:55:38] process terminated successfully, elapsed time: 01.80s
```

Figura 7 – Resultado da verificação Factorial2014

Como se pode observar o GNATprove produz um aviso referente à linha 16 do código,"**factorial2014.adb:16:20: medium: overflow check might fail**". Como o pragma Assert é dado como provado e os limites das iterações estão bem definidos usa-se o pragma Annotate para contornar este erro. A utilização deste pragma requer algum cuidado, pois a sua inclusão no código indica que nunca ocorrerá overflow, apesar do GNATprove não o conseguir provar. A Figura 8 mostra o resultado da verificação após ativação deste, onde se verifica que o erro desapareceu.

```
gnatprove                                                                    -
PD:\_DISSERTACAO\_EXEMPLOS_TESE\Factorial\Factorial2014\factorial2014.gpr
--ide-progress-bar --report=all --level=2
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
factorial2014.adb:7:28: info: initialization of "r" proved
factorial2014.adb:12:13: info: loop invariant preservation proved
factorial2014.adb:12:13: info: loop invariant initialization proved
factorial2014.adb:12:36: info: initialization of "i" proved
```

```
factorial2014.adb:13:23: info: initialization of "i" proved
factorial2014.adb:14:18: info: initialization of "i" proved
factorial2014.adb:14:20: info: overflow check proved
factorial2014.adb:15:22: info: initialization of "r" proved
factorial2014.adb:16:18: info: initialization of "r" proved
factorial2014.adb:16:20: info: range check proved
factorial2014.adb:16:22: info: initialization of "i" proved
factorial2014.adb:18:13: info: assertion proved
factorial2014.adb:18:31: info: initialization of "i" proved
factorial2014.adb:18:42: info: initialization of "r_old" proved
factorial2014.adb:18:48: info: division check proved
factorial2014.adb:18:78: info: overflow check proved
factorial2014.adb:21:16: info: initialization of "r" proved
factorial2014.ads:5:14: info: "Factorial2014" requires body ("Factorial" requires
completion)
factorial2014.ads:9:17: info: postcondition proved
Summary                                  logged                                  in
D:\_DISSERTACAO\_EXEMPLOS_TESE\Factorial\Factorial2014\obj\gnatprove\gnatprove.out
[2016-12-22 16:24:57] process terminated successfully, elapsed time: 01.78s
```

Figura 8 – Resultado da verificação Factorial2014 com pragma Annotate

O SPARK2005 não permite o uso de funções ou procedimentos recursivos pelo facto de a linguagem não permitir alocação dinâmica de memória. Esta caraterística contínua presente na versão atual. Apesar desta restrição, uma das novidades introduzidas pelo SPARK2014 é a possibilidade de utilizar funções recursivas. Isto é possível devido à biblioteca **Formal Containers Library**, que em tempo compilação determina o tamanho necessário de memória e que em tempo de execução aloca o espaço necessário para a função ou procedimento recursivo

A utilização de recursividade deve ser feita com alguma cautela, uma vez que o programador tem de garantir que o procedimento ou função termina. O GNATprove não verifica esta propriedade. O GNATprove trata estas funções como se estas não fossem recursivas, verificando se os contratos estabelecidos na pós-condição são corretos e usa-os para a chamada recursiva.

O código seguinte é a implementação do algoritmo utilizando recursividade. Neste código foi criada uma função **Ghost** para efeitos de prova.

```
001 package FactorialRec2014
002 with SPARK_Mode
003 is
004     function FactorialRec (N : Integer) return Integer
005     with
006         Depends => (FactorialRec'Result => N),
007         pre => (N >= 1 and N <= 12) and (N <= Integer'Last),
008         post =>  (if teste (N) <= Integer'Last and teste (N) = FactorialRec (N)
009            then FactorialRec'Result <= Integer'Last)
010     ;
011     function teste (N : Integer) return Integer
012     with
013         pre => (N >= 1 and N <= 12),
014         post => (teste'result < Integer'Last),
015         Ghost
016     ;
```

```
017
018 end FactorialRec2014;

001 package body FactorialRec2014
002 with SPARK_Mode
003 is
004     function FactorialRec (N : Integer) return Integer
005     is
006     result      : Integer := 1;
007     resultteste : Integer with Ghost;
008     begin
009     resultteste := teste (n);
010
011     if n <= 1 then
012         return 1;
013     end if;
014     if N >= 2 and N <= 12 then
015         result := N * FactorialRec (n - 1);
016     end if;
017     pragma Assert (if teste (N) = FactorialRec (N) and result = resultteste
018         then result <= Integer'Last);
019     return result;
020     end FactorialRec;
021
022     function teste (N : Integer) return Integer
023     is
024     resteste : Integer := 1;
025     begin
026     if N <= 1 then
027         return 1;
028     end if;
029     for I in Integer range N .. 1 loop
030     pragma Loop_Invariant ((N >= 1 and N <= 12) and then
031                 (for all I in N .. 1 =>  I <= 12)
032                 and then (resteste * I <= Integer'Last));
033         resteste := resteste * I;
034         pragma  assert (if I <= N then resteste <= Integer'Last);
035     end loop;
036     return resteste;
037     end teste;
039 end FactorialRec2014;
```

Código 19 – Especificação e implementação da função fatorial recursiva

Como se pode observar, apesar do algoritmo ser simples foi necessário estabelecer vários contratos e uma função de prova para garantir a correção do mesmo. No entanto persiste um erro/aviso como se pode observar na próxima figura com o resultado obtido com o GNATprove.

```
gnatprove -PD:\_DISSERTACAO\DEMO\Factorial\FactorialRec2014\factorialRec2014.gpr --
ide-progress-bar --report=all
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
factorialrec2014.adb:9:24: info: precondition proved
factorialrec2014.adb:15:25: medium: overflow check might fail
factorialrec2014.adb:15:27: info: precondition proved
factorialrec2014.adb:15:43: info: overflow check proved
factorialrec2014.adb:17:09: info: assertion proved
factorialrec2014.adb:17:27: info: precondition proved
factorialrec2014.adb:17:39: info: precondition proved
factorialrec2014.adb:17:60: info: initialization of "result" proved
```

```
factorialrec2014.adb:17:69: info: initialization of "resultteste" proved
factorialrec2014.adb:19:16: info: initialization of "result" proved
factorialrec2014.adb:29:13: info: range check proved
factorialrec2014.adb:30:09: info: loop invariant initialization proved
factorialrec2014.adb:30:09: info: loop invariant preservation proved
factorialrec2014.adb:32:39: info: initialization of "resteste" proved
factorialrec2014.adb:32:48: info: overflow check proved
factorialrec2014.adb:33:25: info: initialization of "resteste" proved
factorialrec2014.adb:33:34: info: overflow check proved
factorialrec2014.adb:34:13: info: assertion proved
factorialrec2014.adb:36:16: info: initialization of "resteste" proved
factorialrec2014.ads:4:14: info: "FactorialRec2014" requires body ("FactorialRec"
requires completion)
factorialrec2014.ads:8:22: info: postcondition proved
factorialrec2014.ads:8:26: info: precondition proved
factorialrec2014.ads:8:56: info: precondition proved
factorialrec2014.ads:8:68: info: precondition proved
factorialrec2014.ads:11:14: info: "FactorialRec2014" requires body ("teste" requires
completion)
factorialrec2014.ads:14:21: info: postcondition proved
Summary                                  logged                                  in
D:\_DISSERTACAO\DEMO\Factorial\FactorialRec2014\obj\gnatprove\gnatprove.out
[2017-01-22 23:45:01] process terminated successfully, elapsed time: 00.89s
```

Figura 9 – Resultado da verificação Factorial2014Rec

A utilização da função teste tem o intuito de fortalecer a demonstração da correção do algoritmo, no entanto persiste o erro/aviso assinalado na Figura 9. Há necessidade de rever os contratos de forma a eliminar este erro/aviso sem recorrer ao pragma Annotate.

5.4 N-Queens

No prolema das N-Queens, pretende-se dispor N Rainhas num tabuleiro de xadrez com dimensões N × N, para que nenhuma destas se ataque. Este problema é utilizado no ensino de algoritmia. Embora este problema possa ser resolvido de forma iterativa, a solução recursiva envolve menos código, tornando-a mais elegante.

O algoritmo que se apresenta também recorre à recursividade para a resolução do problema N-Queens. Este código só apresenta uma solução e está correta, no entanto, a especificação dos contratos é difícil e neste caso está incompleta.

Na Figura 10 pode-se observar que o GNATprove não consegue provar que o código está livre de erros, ou seja, é necessário rever e/ou alterar os contratos estabelecidos.

```
001 package NQueens2014
002 with SPARK_Mode
003
004 is
005     type Tabuleiro is  array (Positive range <>, Positive range <>) of Boolean;
```

```
006     Tab : Tabuleiro (1 .. 9, 1 .. 9) := (others => (others => False));
007
008     function Testa (Linha, Coluna : Integer) return Boolean
009     with
010     pre =>((Linha >= 1 and Linha <= 9) and (Coluna >= 0 and Coluna <= 0))
011     ;
012
013     procedure Preenche (Coluna : Integer; Result : out Boolean)
014     with
015     pre => (Coluna >= 1 and Coluna <= 9),
016     post => (for all L in 1..9 => (for all C in 1..9 => Tab(L,C) = Testa(L,C)))
017     ;
018
019 end NQueens2014;
```

```
001 package body NQueens2014
002 with SPARK_Mode
003
004 is
005     function Testa (Linha, Coluna : Integer) return Boolean is
006     begin
007     for J in 1 .. Coluna - 1 loop
008         pragma Loop_Invariant (Coluna >= 1 and Coluna <= 9);
009         if (Tab (Linha, J)
010         or else
011         (Linha > J and then Tab (Linha - J, Coluna - J))
012         or else
013         (Linha + J <= 9 and then Tab (Linha + J, Coluna - J)))
014         then
015         return False;
016         end if;
017     end loop;
018     return True;
019     end Testa;
020
021     procedure Preenche (Coluna : Integer; Result : out Boolean) --return Boolean
022     is
023     begin
024     for Linha in Tab'Range (1) loop
025         pragma Loop_Invariant (Linha >= Tab'First and Linha <= Tab'Last);
026         if Testa (Linha, Coluna) then
027         Tab (Linha, Coluna) := True;
028         if Coluna = 9 then
029             Result := True;
030             return;
031         else
032             Preenche (Coluna + 1, Result);
033             if Result = True then
034             return;
035             end if;
036         end if;
037         Tab (Linha, Coluna) := False;
038         Result := False;
039         end if;
040     end loop;
041
042     end Preenche;
043
044 end Nqueens2014;
```

Código 20 – Especificação e implementação de NQueens

```
gnatprove  -PD:\_DISSERTACAO\DEMO\N_QUEENS\n_queens.gpr  --ide-progress-bar  --report=all  --
level=1
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
```

```
nqueens2014.adb:7:30: medium: overflow check might fail (e.g. when Coluna = -2147483648)
nqueens2014.adb:9:22: medium: array index check might fail (e.g. when Linha = 0)
nqueens2014.adb:9:29: medium: array index check might fail (e.g. when J = 10)
nqueens2014.adb:11:48: info: index check proved
nqueens2014.adb:11:48: info: overflow check proved
nqueens2014.adb:11:60: medium: array index check might fail
nqueens2014.adb:11:60: info: overflow check proved
nqueens2014.adb:13:24: info: overflow check proved
nqueens2014.adb:13:53: info: index check proved
nqueens2014.adb:13:53: info: overflow check proved
nqueens2014.adb:13:65: medium: array index check might fail
nqueens2014.adb:13:65: info: overflow check proved
nqueens2014.adb:27:29: medium: array index check might fail (e.g. when Coluna = 0)
nqueens2014.adb:32:38: info: overflow check proved
nqueens2014.adb:33:24: info: initialization of "Result" proved
nqueens2014.adb:37:29: info: index check proved
nqueens2014.ads:6:41: info: length check proved
nqueens2014.ads:13:43: medium: "Result" might not be initialized in "Preenche"
Summary logged in D:\_DISSERTACAO\DEMO\N_QUEENS\gnatprove\gnatprove.out
[2017-02-01 22:05:39] process terminated successfully, elapsed time: 06.52s
```

Figura 10 – Resultado da verificação N-Queens

Com a adição de contratos simples na especificação (linhas 10 e 14) e na implementação (linhas 8 e 27), os erros reportados anteriormente e presentes na Figura 10 são eliminados com esta nova versão.

```
001     package NQueens2014
002     with SPARK_Mode
003
004     is
005             type Tabuleiro is  array (Positive range <>, Positive range <>) of Boolean;
006             Tab : Tabuleiro (1 .. 9, 1 .. 9) := (others => (others => False));
007
008             function Testa (Linha, Coluna : Integer) return Boolean
009             with
010                     pre => ((Linha >= 1 and Linha <= 9) and (Coluna >= 1 and Coluna <= 9))
011             ;
012
013             procedure Preenche (Coluna : Integer; Result : out Boolean)
014             with
015                     pre => (Coluna >= 1 and Coluna <= 9)
016             ;
017
018     end NQueens2014;
```

```
001     package body NQueens2014
002     with SPARK_Mode
003
004     is
005             function Testa (Linha, Coluna : Integer) return Boolean is
006             begin
007             for J in 1 .. Coluna - 1 loop
008                     pragma Loop_Invariant (Coluna >= 1 and Coluna <= 9);
009                     if (Tab (Linha, J)
010                     or else
011                     (Linha > J and then Tab (Linha - J, Coluna - J))
012                     or else
013                     (Linha + J <= 9 and then Tab (Linha + J, Coluna - J)))
014                     then
015                     return False;
016                     end if;
017             end loop;
018             return True;
019             end Testa;
```

```
020
021             procedure Preenche (Coluna : Integer; Result : out Boolean)
022             is
023             begin
024             Result := False;
025
026             for Linha in Tab'Range (1) loop
027                     pragma Loop_Invariant (Linha >= Tab'First (1) and Linha <= Tab'Last
(1));
028                     if Testa (Linha, Coluna) then
029                     Tab (Linha, Coluna) := True;
030                     if Coluna = 9 then
031                             Result := True;
032                             return;
033                     else
034                             Preenche (Coluna + 1, Result);
035                             if Result = True then
036                             return;
037                             end if;
038                     end if;
039                     Tab (Linha, Coluna) := False;
040                     Result := False;
041                     end if;
042             end loop;
043
044             end Preenche;
045
046     end Nqueens2014;
```

Código 21 – Versão de NQueens com contratos

```
gnatprove  -PD:\_DISSERTACAO\DEMO\N_QUEENS\nqueens2014.gpr  --level=0  --ide-progress-bar  --
report=all
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
nqueens2014.adb:7:30: info: overflow check proved
nqueens2014.adb:8:13: info: loop invariant preservation proved
nqueens2014.adb:8:13: info: loop invariant initialization proved
nqueens2014.adb:9:22: info: index check proved
nqueens2014.adb:9:29: info: index check proved
nqueens2014.adb:11:48: info: index check proved
nqueens2014.adb:11:48: info: overflow check proved
nqueens2014.adb:11:60: info: overflow check proved
nqueens2014.adb:11:60: info: index check proved
nqueens2014.adb:13:24: info: overflow check proved
nqueens2014.adb:13:53: info: index check proved
nqueens2014.adb:13:53: info: overflow check proved
nqueens2014.adb:13:65: info: index check proved
nqueens2014.adb:13:65: info: overflow check proved
nqueens2014.adb:26:13: info: loop invariant preservation proved
nqueens2014.adb:26:13: info: loop invariant initialization proved
nqueens2014.adb:27:16: info: precondition proved
nqueens2014.adb:28:29: info: index check proved
nqueens2014.adb:33:21: info: precondition proved
nqueens2014.adb:33:38: info: overflow check proved
nqueens2014.adb:34:24: info: initialization of "Result" proved
nqueens2014.adb:38:29: info: index check proved
nqueens2014.ads:6:41: info: length check proved
nqueens2014.ads:13:43: info: initialization of "Result" proved
Summary logged in D:\_DISSERTACAO\DEMO\N_QUEENS\gnatprove\gnatprove.out
[2017-03-11 13:20:22] process terminated successfully, elapsed time: 02.53s
```

Figura 11 – Resultado da verificação da nova versão NQueens

O algoritmo que se segue, Código 22, resolve o problema N-Queens de forma iterativa e apresenta as soluções fundamentais para tabuleiros 1x1 até 20x20. É uma versão mais complexa que a anterior e a formulação das anotações para prova, mais difícil. O código apresentado tem os contratos comentados.

```
001     with GNAT.IO; use GNAT.IO;
002
003     package N_Queens2014
004     with SPARK_Mode
005     is
006             Max_Nrqueens     : constant := 20;
007             MAx_Sol_Nrqueens : constant := 39029188884;
008             ctd              : Long_Long_Integer := 0;
009             NrQueens         : Natural := Max_Nrqueens;
010
011             contador : Integer := 0;
012
013             type Tabuleiro_tipo is array (1 .. NrQueens, 1 .. NrQueens)of Natural range
0 .. 1;
014             Tabuleiro : Tabuleiro_tipo;
015             type Apontador_tipo is array (1 .. NrQueens) of Natural range 0 .. Max_Nrqueens
+ 1;
016             Apontador : Apontador_tipo;
017
018
019
020             function TestaSolucao (Tabuleiro : in Tabuleiro_tipo) return Boolean
021     --      with
022     --          pre => (Tabuleiro'First (1) = 1 and Tabuleiro'Last (1) = NrQueens
023     --           and Tabuleiro'First (2) = 1 and Tabuleiro'Last (2) = NrQueens)
024     --          and (Tabuleiro'First (1) = 1 and Tabuleiro'Last (1) = NrQueens - 1
025     --           and Tabuleiro'First (2) = 1 and Tabuleiro'Last (2) = NrQueens)
026             ;
027
028
029             procedure NQueens (N : in Natural)
030     --      with
031     --          pre => (N > 0 and N < Max_Nrqueens)
032     --      and (Apontador'First = 1 and Apontador'Last = NrQueens),
033     --      post => (for all i in Tabuleiro'Range (1) =>
034     --                 (for all j in Tabuleiro'Range (2) => Tabuleiro (i, j) < 2))
035     --          and (for all j in Apontador'Range => Apontador (j) >= 0
036     --           and Apontador (j) <= (Max_Nrqueens + 1))
037             ;
038
039     end N_Queens2014;


001     with GNAT.IO; use GNAT.IO;
002     with PrintSolution;
003
004     package body N_Queens2014
005     with SPARK_Mode
006     is
007             function TestaSolucao (Tabuleiro : in Tabuleiro_tipo) return Boolean
008             is
009             soma, p, q : Integer;
010             idx_J      : Integer;
011
012             begin
013
014             for i in 1 .. Nrqueens loop
015                     soma := 1;
016                     for j in 1 .. Nrqueens loop
```

```
017                     soma := soma + Tabuleiro (i, j);
018                     if soma > 2 then
019                             return False;
020                     end if;
021                     end loop;
022             end loop;
023
024
025             -- verifica diagonal
026             idx_J := NrQueens - 1;
027             while idx_J >= 1 loop
028                     soma := 1;
029                     p := 1;
030                     q := idx_J;
031                     while q <= NrQueens loop
032                     soma := soma + Tabuleiro (p, q);
033                     p := p + 1;
034                     q := q + 1;
035                     end loop;
036                     if soma > 2 then
037                     return False;
038                     end if;
039                     idx_J := idx_J - 1;
040             end loop;
041             for i in Integer range 2 .. NrQueens - 1 loop
042                     soma := 1;
043                     p := i;
044                     q := 1;
045                     while p <= NrQueens loop
046                     soma := soma + Tabuleiro (p, q);
047                     p := p + 1;
048                     q := q + 1;
049                     end loop;
050                     if soma > 2 then
051                     return False;
052                     end if;
053             end loop;
054
055             -- verifica diagonal
056             for j in Integer range 2 .. NrQueens loop
057                     soma := 1;
058                     p := 1;
059                     q := j;
060                     while q >= 1 loop
061                     soma := soma + Tabuleiro (p, q);
062                     p := p + 1;
063                     q := q - 1;
064                     end loop;
065                     if soma > 2 then
066                     return False;
067                     end if;
068             end loop;
069
070             for i in Integer range 2 .. NrQueens - 1  loop
071                     soma := 1;
072                     p := i;
073                     q := NrQueens;
074                     while p <= NrQueens loop
075                     soma := soma + Tabuleiro (p, q);
076                     p := p + 1;
077                     q := q - 1;
078                     end loop;
079                     if soma > 2 then
080                     return False;
081                     end if;
082             end loop;
083             return true;
084             end TestaSolucao;
085
086
087
```

```
088             procedure NQueens (N : in Natural)
089             is
090             ind : Integer := 1;
091             begin
092             ctd := 0;
093             NrQueens := N;
094             pragma Assert (N <= Max_Nrqueens);
095
096             while ind <= NrQueens loop
097                     pragma Loop_Invariant (ind > 0 and ind <= Nrqueens);
098                     Apontador (ind) := Apontador (ind) + 1;
099     --          pragma Annotate (GNATprove, Intentional,
100     --                   "", "ind e sempre <= Nrquens");
101                     if Apontador (ind) = NrQueens + 1 then
102                     Tabuleiro (Apontador (ind)-1, ind) := 0;
103     --              pragma Annotate (GNATprove, Intentional,
104     --                 "", "Apontador (ind)-1 e sempre >= 1");
105                     Apontador (ind) := 0;
106                     ind := ind - 1;
107                     if ind = 0 then
108                             return;
109                     end if;
110                     else
111                     Tabuleiro (Apontador (ind), ind) := 1;
112     --              pragma Annotate (GNATprove, Intentional,
113     --                 "", "Apontador (ind)-1 e sempre >= 1");
114                     if Apontador (ind) /= 1 then
115                             Tabuleiro (Apontador (ind) - 1, ind) := 0;
116                     end if;
117                     if TestaSolucao (Tabuleiro) then
118     --                  pragma Annotate (GNATprove, Intentional,
119     --                     "", "Apontador (ind)-1 e sempre >= 1");
120                             ind := ind + 1;
121                             if ind = NrQueens + 1 then
122                             ind := ind - 1;
123     --                      pragma  Assume  (ctd  <  Max_Sol_Nrqueens,  "ctd  e  sempre  <
Long_Long_Long_Integer");
124                             ctd := ctd + 1;
125                             PrintSolution (ctd, Nrqueens, Tabuleiro);
126                             end if;
127                     end if;
128                     end if;
129                     exit when ind > NrQueens;
130             end loop;
131
132             end NQueens;
133
134     end N_Queens2014;
```

Código 22 – Versão iterativa de N-Queens com contratos comentados

Na Figura 12 apresentam-se os resultados do GNATProve sobre o código tal como está presente em Código 22.

```
Gnatprove –PD:\_DISSERTACAO\_EXEMPLOS_TESE\N_Queens\n_queens2014.gpr –level=0 –ide-progress-
bar
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
n_queens2014.adb:17:30: medium: overflow check might fail
n_queens2014.adb:17:43: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 0 and Tabuleiro'Last(2) = 1 and i = 1)
n_queens2014.adb:17:46: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 0 and Tabuleiro'Last(2) = 0 and j = 1)
n_queens2014.adb:32:30: medium: overflow check might fail
n_queens2014.adb:32:43: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 2 and Tabuleiro'Last(2) = 2 and p = 3)
n_queens2014.adb:32:46: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 2 and Tabuleiro'Last(2) = 2 and q = 0)
n_queens2014.adb:33:24: medium: overflow check might fail (e.g. when p = 2147483647)
n_queens2014.adb:34:24: medium: overflow check might fail (e.g. when q = 2147483647)
n_queens2014.adb:46:30: medium: overflow check might fail
n_queens2014.adb:46:43: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 3 and Tabuleiro'Last(2) = 3 and p = 0)
n_queens2014.adb:46:46: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 3 and Tabuleiro'Last(2) = 3 and q = 4)
n_queens2014.adb:48:24: medium: overflow check might fail (e.g. when q = 2147483647)
n_queens2014.adb:61:30: medium: overflow check might fail
n_queens2014.adb:61:43: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 2 and Tabuleiro'Last(2) = 2 and p = 3)
n_queens2014.adb:61:46: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 2 and Tabuleiro'Last(2) = 2 and q = 3)
n_queens2014.adb:62:24: medium: overflow check might fail (e.g. when p = 2147483647)
n_queens2014.adb:75:30: medium: overflow check might fail
n_queens2014.adb:75:43: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 3 and Tabuleiro'Last(2) = 3 and p = 0)
n_queens2014.adb:75:46: medium: array index check might fail (e.g. when Tabuleiro'First(1) = 1
and Tabuleiro'First(2) = 1 and Tabuleiro'Last(1) = 3 and Tabuleiro'Last(2) = 3 and q = 4)
n_queens2014.adb:94:24: medium: assertion might fail, cannot prove N <= Max_Nrqueens (e.g. when
N = 21)
n_queens2014.adb:98:43: medium: array index check might fail (e.g. when Apontador'Last = 0 and
ind = 1)
n_queens2014.adb:98:48: medium: range check might fail
n_queens2014.adb:102:43: medium: array index check might fail (e.g. when Tabuleiro'Last (1) =
0)
n_queens2014.adb:102:47: medium: array index check might fail (e.g. when Tabuleiro'Last (2) =
0 and ind = 1)
n_queens2014.adb:111:28: medium: array index check might fail (e.g. when Tabuleiro'Last (1) =
0)
n_queens2014.adb:111:45: medium: array index check might fail (e.g. when Tabuleiro'Last (2) =
0 and ind = 1)
n_queens2014.adb:124:36:   medium:   overflow   check   might   fail   (e.g.   when   ctd   =
9223372036854775807)
Summary logged in D:\_DISSERTACAO\_EXEMPLOS_TESE\N_Queens\obj\gnatprove\gnatprove.out
[2017-05-01 12:57:37] process terminated successfully, elapsed time: 01.17s
```

Figura 12 – Resultado da verificação da versão NQueens iterativa

Como se pode observar são produzidos 27 erros. A figura seguinte, Figura 13, apresenta o resultado obtido após a ativação das anotações nas linhas 21 a 25 e 30 a 36 da especificação.

```
gnatprove -PD:\_DISSERTACAO\_EXEMPLOS_TESE\N_Queens\n_queens2014.gpr --level=0 --ide-progress-
bar
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
n_queens2014.adb:98:43: medium: array index check might fail (e.g. when Apontador'Last = 0 and
ind = 1)
n_queens2014.adb:98:48: medium: range check might fail
n_queens2014.adb:102:43: medium: array index check might fail (e.g. when Tabuleiro'Last (1) =
0)
n_queens2014.adb:102:47: medium: array index check might fail (e.g. when Tabuleiro'Last (2) =
0 and ind = 1)
n_queens2014.adb:111:28: medium: array index check might fail (e.g. when Tabuleiro'Last (1) =
0)
n_queens2014.adb:111:45: medium: array index check might fail (e.g. when Tabuleiro'Last (2) =
0 and ind = 1)
n_queens2014.adb:117:20: medium: precondition might fail (e.g. when NrQueens = 1 and
Tabuleiro'First (1) = 1 and Tabuleiro'First (2) = 1 and Tabuleiro'Last (1) = 3 and Tabuleiro'Last
(2) = 1)
n_queens2014.adb:124:36: medium: overflow check might fail (e.g. when ctd =
9223372036854775807)
Summary logged in D:\_DISSERTACAO\_EXEMPLOS_TESE\N_Queens\obj\gnatprove\gnatprove.out
[2017-05-01 23:10:19] process terminated successfully, elapsed time: 12.78s
```

Figura 13 – Resultado da verificação da versão NQueens iterativa

O número de erros apresentado diminuiu consideravelmente, no entanto, embora o código esteja correto, a escrita de contratos para provar formalmente a sua correção é difícil. Usando várias anotações/contratos não se conseguiu resolver estes erros, optando-se por incluir os pragmas constantes nas linhas 99, 103, 112, 118, e 123 da implementação. A utilização deste tipo de anotações requer cautela e moderação, só devendo ser usado em situações que haja a certeza as mensagens de erro geradas ferramenta de prova não são efetivamente erros de programa.

5.5 Produtor/Consumidor

O modelo Produtor/Consumidor é usado normalmente para a demonstração de concorrência. O exemplo presente em Código 23 é muito simples e não foram escritos quaisquer contratos para verificação, com exceção da utilização dos pragmas Profile(Ravenscar) [33] Partition_Elaboration_Policy. Estes pragmas são necessários para que o SPARK permita a execução de código concorrente.

```
gnatprove   -PD:\_DISSERTACAO\DEMO\Prod_Cons_SPARK2014   _OLD\prod_cons_spark2014.gpr   --ide-
progress-bar --report=all --level=1
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
prod_cons_spark2014.adb:8:05: tasking in SPARK requires Ravenscar profile (SPARK RM 9(2))
prod_cons_spark2014.adb:8:05: violation of aspect SPARK_Mode at line 6
prod_cons_spark2014.adb:17:05: tasking in SPARK requires Ravenscar profile (SPARK RM 9(2))
prod_cons_spark2014.adb:17:05: violation of aspect SPARK_Mode at line 6
prod_cons_spark2014.adb:28:05: tasking in SPARK requires Ravenscar profile (SPARK RM 9(2))
prod_cons_spark2014.adb:28:05: violation of aspect SPARK_Mode at line 6
prod_cons_spark2014.ads:9:19: tasking in SPARK requires Ravenscar profile (SPARK RM 9(2))
prod_cons_spark2014.ads:9:19: violation of aspect SPARK_Mode at line 7
prod_cons_spark2014.ads:20:06: tasking in SPARK requires Ravenscar profile (SPARK RM 9(2))
prod_cons_spark2014.ads:20:06: violation of aspect SPARK_Mode at line 7
prod_cons_spark2014.ads:21:06: tasking in SPARK requires Ravenscar profile (SPARK RM 9(2))
prod_cons_spark2014.ads:21:06: violation of aspect SPARK_Mode at line 7
gnatprove: error during flow analysis and proof
[2017-02-05 23:44:06] process exited with status 1, 50% (1/2), elapsed time: 00.45s
```

Figura 14 – Resultado da verificação Prod_Cons_SPARK2014 sem pragmas

Como se pode observar na Figura 14, a verificação do código sem a utilização dos pragmas atrás referenciados origina vários erros.

O pragma Profile(Ravenscar) define um subconjunto de regras/características de concorrência da linguagem Ada, tendo como principal foco a analise/verificação do agendamento de tarefas e uso de memória. Permite também a sincronização e comunicação entre tarefas.

O pragma Partition_Elaboration_Policy garante que as tarefas não se iniciam antes de o programa estar completamente em memória e em execução.

```
001 pragma Profile (Ravenscar);
002 pragma Partition_Elaboration_Policy (Sequential);
003
004 with Ada.Text_IO; use Ada.Text_IO;
005
006 package Prod_Cons_SPARK2014
007 with SPARK_Mode
008 is
009    protected type Buffer is
010       procedure PutValue (Val : in Integer);
011       function GetValue return Integer;
012     private
013     Num : Integer := 0;
014    end Buffer;
015
016    Nr : Buffer;
017    Nr_Max : constant Natural := 100;
018
019
020 task Producer;
021 task Consumer;
022
023 end Prod_Cons_SPARK2014;
```

```
001 pragma Profile (Ravenscar);
002 pragma Partition_Elaboration_Policy (Sequential);
003 with Ada.Text_IO; use Ada.Text_IO;
004
005 package body Prod_Cons_SPARK2014
006 With SPARK_Mode
007 is
008     protected body Buffer is
009     procedure PutValue (Val : in Integer) is
```

```
010     begin
011         Num := Val;
012     end PutValue;
013
014     function GetValue return Integer is (Num);
015     end Buffer;
016
017     task body Producer is
018     begin
019     for I in 1 .. 1_000_000 loop
020         put ("PUT: ");
021         Put_Line
022         (Integer'Image (I));
023         Nr.PutValue (I);
024     end loop;
025     end Producer;
026
027
028     task body Consumer is
029     My_Value : Integer;
030     begin
031     loop
032          My_Value:= Nr.GetValue;
033         Put_Line
034         (Integer'Image (My_Value));
035     end loop;
036     end Consumer;
037
038 begin
039     null;
040 --end;
041 end Prod_Cons_SPARK2014;
```

Código 23 – Especificação e implementação de Prod_Cons_SPARK2014

Como se pode observar, na Figura 15, o resultado da verificação código usando os pragmas, gera vários avisos e um erro. O erro informa que não existe a garantia que a tarefa não termine, condição que não é permitida pelo SPARK.

```
gnatprove   -PE:\_DISSERTACAO\DEMO\Prod_Cons_SPARK2014   _OLD\prod_cons_spark2014.gpr   --ide-
progress-bar --report=all -j0
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
prod_cons_spark2014.adb:20:13: warning: no Global contract available for "Put"
prod_cons_spark2014.adb:20:13: warning: assuming "Put" has no effect on global items
prod_cons_spark2014.adb:21:13: warning: no Global contract available for "Put_Line"
prod_cons_spark2014.adb:21:13: warning: assuming "Put_Line" has no effect on global items
prod_cons_spark2014.adb:33:13: warning: no Global contract available for "Put_Line"
prod_cons_spark2014.adb:33:13: warning: assuming "Put_Line" has no effect on global items
prod_cons_spark2014.adb:34:33: info: initialization of "My_Value" proved
prod_cons_spark2014.ads:20:06: medium: the task might terminate, which is not allowed in SPARK
prod_cons_spark2014.ads:20:06: info: ceiling priority protocol is respected
prod_cons_spark2014.ads:21:06: info: nontermination of task proved
prod_cons_spark2014.ads:21:06: info: ceiling priority protocol is respected
Summary logged in E:\_DISSERTACAO\DEMO\Prod_Cons_SPARK2014 _OLD\gnatprove\gnatprove.out
[2017-02-06 18:47:34] process terminated successfully, elapsed time: 05.23s
```

Figura 15 – Resultado da verificação Prod_Cons_SPARK2014 utilizando pragmas

O código necessita de ser revisto e anotado, para que sejam respeitadas as regras da linguagem referentes à concorrência e às tarefas. O código que se segue, Código 24, foi alterado de forma a garantir a ausência de erros.

```
001     pragma Profile (Ravenscar);
002     pragma Partition_Elaboration_Policy (Sequential);
003
004     package Prod_Cons
005     with SPARK_Mode
006     is
007   numero : Integer := 0;
008
009   protected type Buffer is
010      procedure PutValue (Val : in Integer)
011     ;
012      function GetValue return Integer
013     ;
014   private
015      Num : Integer := 0;
016   end Buffer;
017
018   Nr     : Buffer;
019
020   task Producer
021     with
022     Global   => (In_Out => Nr),
023     Depends  => (Nr => Nr)
024    ;
025
026   task Consumer
027     with
028     Global   => (In_Out => Nr),
029     Depends  => (Nr => Nr)
030    ;
031
032 end Prod_Cons;
```

```
001     pragma Profile (Ravenscar);
002     pragma Partition_Elaboration_Policy (Sequential);
003     with Gnat.IO; use GNAT.IO;
004
005     package body Prod_Cons
006     With SPARK_Mode
007     is
008
009   protected body Buffer is
010      procedure PutValue (Val : in Integer) is
011      begin
012      Num := Val;
013      end PutValue;
014
015      function GetValue return Integer is (Num);
016
017   end Buffer;
018
019
020   task body Producer is
021   begin
022      loop
023      if numero < Integer'Last then
024         numero := numero + 1;
025      else
026         numero := 0;
027      end if;
028      Nr.PutValue (numero);
029      Put_Line ("PUT: " & Integer'Image (numero));
030      pragma Annotate (GNATprove, Intentional,
031                "", "Put_Line foi adicionado somente p/mostrar valores. A retirar");
032      end loop;
033   end Producer;
034
035
036   task body Consumer is
037      My_Value     : Integer;
038      My_Value_Old : Integer := 0;
```

```
039   begin
040      loop
041      My_Value := Nr.GetValue;
042      if My_Value_Old /= My_Value then
043         My_Value_Old := My_Value;
044         Put_Line ("GET: " & Integer'Image (My_Value) & " <<-----------");
045         pragma Annotate (GNATprove, Intentional,
046                          "", "Put_Line foi adicionado somente p/mostrar valores. A retirar");
047      end if;
048      end loop;
049   end Consumer;
050
051 end Prod_Cons;
```

Código 24 – Especificação e implementação de nova versão Prod_Cons_SPARK2014

Com as alterações introduzidas, o erro que indicava que a tarefa podia terminar desaparece mantendo-se somente os avisos referentes à utilização do procedimento **Put_Line.** Este procedimento é usado somente para teste, devendo ser retirado. Estes avisos devem-se ao facto de a ferramenta GNATProve não gerar contratos para a biblioteca standard **GNAT.IO** utilizada no código, que não implementa contratos. Estes avisos podem ser eliminados usando o switch **--warnings=off** com o GNATProve, no entanto, estes poderão ser uteis numa fase inicial de desenvolvimento pelo que não se aconselha a sua desativação. Outra forma forma, mais elaborada, de eliminar estes avisos será construir uma biblioteca baseada em **GNAT.IO** com a uma especificação em SPARK.

```
gnatprove  -PE:\_DISSERTACAO\_EXEMPLOS_TESE\Prod_Cons_VersaodaTese_sem_stack\prod_cons.gpr  --
ide-progress-bar
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
prod_cons.adb:29:13: warning: no Global contract available for "Put_Line"
prod_cons.adb:29:13: warning: assuming "Put_Line" has no effect on global items
prod_cons.adb:44:17: warning: no Global contract available for "Put_Line"
prod_cons.adb:44:17: warning: assuming "Put_Line" has no effect on global items
Summary                                    logged                                        in
E:\_DISSERTACAO\_EXEMPLOS_TESE\Prod_Cons_VersaodaTese_sem_stack\obj\gnatprove\gnatprove.out
[2017-03-01 17:02:14] process terminated successfully, elapsed time: 06.54s
```

Figura 16 – Resultado da verificação da nova versão deProd_Cons_SPARK2014

Como foi referido anteriormente a concorrência em SPARK tem algumas restrições impostas pelo perfil Ravenscar. Este perfil foi definido com o objetivo de responder às necessidades de concorrência em aplicações de tempo real, aplicações que exigem certificação formal, aplicações que exigem verificação formal e aplicações para sistemas embebidos que normalmente têm recursos limitados. Esta imposição permite a execução de código concorrente que pode ser analisado e provado pela ferramenta GNATProve. A comunicação entre tarefas, **tasks,** é feita através de objetos partilhados sincronizados, de forma a garantir que não haja acesso simultâneo por várias tarefas. Esta garantia é dada pela utilização de **protected objects**, **suspension objects**, **atomic objects**, **constants**.

5.6 Stack concorrente

As variantes de código que se seguem, adaptação do exemplo Pilha, implementam 3 tarefas/task, 2 de Push e uma de Pop que comunicam através do objecto protegido StackPC.

```
001     -- SPARK 2014
002     pragma SPARK_Mode;
003     pragma Profile (Ravenscar);
004     pragma Partition_Elaboration_Policy (Sequential);
005
006
007     package Stack
008     is
009
010        Stack_Size : constant := 5;
011        type Pointer_Stack is range 0 .. Stack_Size;
012        subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
013        type Vector is array (Index_Stack) of Integer;
014        Size_ini   : Integer := 0;
015
016
017
018        protected type StackPC is
019               function Empty return Boolean
020             ;
021               function Full return Boolean
022             ;
023               procedure Push (I : in Integer)
024             ;
025               procedure Pop (I : out Integer)
026             ;
027        private
028               Size_of_Stack : Pointer_Stack := 0;
029               Data_in_Stack : Vector := Vector'(Index_Stack => 0);
030               Not_Full      : Boolean := True;
031        end StackPC;
032
033
034        StackTask : StackPC;
035
036        task type PushStack
037              with
038                Global => (In_Out => Stacktask),
039              Depends  => (PushStack => PushStack,
040                           StackTask => StackTask)
041               ;
042        task PopStack
043              with
044                Global => (In_Out => Stacktask),
045              Depends  => (PopStack  => PopStack,
046                           StackTask => StackTask)
047             ;
048
049        task PushStack1
050              with
051                Global => (In_Out => Stacktask),
052              Depends  => (PushStack1 => PushStack1,
053                           StackTask  => StackTask)
054               ;
055
056     end Stack;
```

```
001     -- SPARK 2014
002     pragma SPARK_Mode;
003     pragma Profile (Ravenscar);
004     pragma Partition_Elaboration_Policy (Sequential);
005
006     with Gnat.Io; use Gnat.Io;
```

```
007     with PrintPopPush;
008
009     package body Stack
010     is
011        protected body StackPC
012        is
013               function Full return Boolean is (Size_of_Stack = Stack_Size);
014
015               function Empty return Boolean is (Size_of_Stack = 0);
016
017               procedure Push (I : in Integer)
018               is
019               begin
020              Size_of_Stack := Size_of_Stack + 1;
021              Data_in_stack (Size_of_stack) := I;
022              if I >= 5000000 then
023                     PrintPopPush   ("PUSH",   "PushStack   -   Valor:   "   &   Integer'Image
(Data_in_stack (Size_of_stack)) & " #########");
024              else
025                     PrintPopPush   ("PUSH",   "PushStack   -   Valor:   "   &   Integer'Image
(Data_in_stack (Size_of_stack)) & " <><><><><>");
026              end if;
027              pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para
efeitos de teste");
028               end Push;
029
030               procedure Pop (I : out Integer)
031               is
032               begin
033              I := Data_in_stack (Size_of_stack);
034              Size_of_Stack := Size_of_Stack - 1;
035              PrintPopPush ("POP", "PopStack - Valor: " & Integer'Image (I) & " ----------
------------------");
036              pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para
efeitos de teste");
037              PrintPopPush  ("SIZE",  "PopStack  -  SizeStack:  "  &  Integer'Image  (Integer
(Size_of_Stack)) & "");
038              pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para
efeitos de teste");
039               end Pop;
040
041        end StackPC;
042
043        task body PushStack
044        is
045               nr     : Integer := 0;
046               T_Full : Boolean := False;
047        begin
048               loop
049              T_Full := StackTask.Full;
050              if  not T_Full then
051                     if nr < Integer'Last then
052                        nr := nr + 1;
053                     else
054                        nr := 1;
055                     end if;
056                     StackTask.Push (nr);
057              end if;
058               end loop;
059        end PushStack;
060
061        task body PopStack is
062               nr      : Integer := 0;
063               T_Empty : Boolean := False;
064        begin
065               loop
066              T_Empty := StackTask.Empty;
067              if not T_Empty then
068                     StackTask.Pop (nr);
069              end if;
070
```

```
071               end loop;
072         end PopStack;
073
074         task body PushStack1
075         is
076                nr     : Integer := 5000000;
077                T_Full : Boolean := False;
078         begin
079                loop
080               T_Full := StackTask.Full;
081               if  not T_Full then
082                      if nr < Integer'Last then
083                         nr := nr + 1;
084                      else
085                         nr := 5000000;
086                      end if;
087                      StackTask.Push (nr);
088               end if;
089                end loop;
090         end PushStack1;
091
092
093      end Stack;
```

Código 25 – Stack concorrente usando objecto protegido StackPC

```
gnatprove   -PE:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese\stack_prod_cons.gpr   --ide-
progress-bar
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
stack.adb:20:44: medium: range check might fail
stack.adb:34:33: medium: array index check might fail
stack.ads:18:20: info: "Stack" requires body ("StackPC" requires completion)
stack.ads:23:19: high: potentially blocking operation in protected operation "Push"
stack.ads:25:19: high: potentially blocking operation in protected operation "Pop"
stack.ads:36:15: info: "Stack" requires body ("PushStack" requires completion)
stack.ads:42:10: info: "Stack" requires body (type derived from "PopStack" requires completion)
stack.ads:49:10:  info:  "Stack"  requires  body  (type  derived  from  "PushStack1"  requires
completion)
Summary                                        logged                                          in
E:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese\obj\gnatprove\gnatprove.out
[2017-05-13 18:56:46] process terminated successfully, elapsed time: 05.30s
```

Figura 17 – Resultado da verificação de Stack concorrente (Código 25)

Como se pode verificar são identificados 4 erros. Os 2 erros que indicam a existência de um potencial bloqueio das operações Push e Pop, são devidos ao código PrintPopPush que foi criado somente para efeitos de teste e que deve ser retirado. Os outros 2 acontecem porque não existe qualquer controlo sobre o valor do índice do vetor Data_in_stack. A verificação se a Pilha está cheia ou vazia é feita no corpo das tarefas, o que pode conduzir a uma condição de corrida, **race condition**. No caso das tarefas/tasks PushStack e PushStack1, ambas podem obter o mesmo estado da Pilha, não cheia. Embora neste exemplo se esteja a usar um objecto protegido, procedimentos e funções protegidas, no espaço de tempo que medeia a verificação do estado e adição de um novo valor à pilha por parte de uma tarefa, pode permitir que a outra tarefa verifique o estado e obtenha o mesmo valor. Se após a adição do valor a Pilha ficar cheia, a adição de novo valor por parte da outra tarefa, vai provocar a interrupção do programa ao

incrementar o valor do índice do vetor ultrapassando o limite superior do mesmo. Esta situação não se verifica com a tarefa PopStack, porque só existe uma tarefa a retirar elementos da Pilha.

Para evitar a situação atrás descrita, Código 25, foi feita uma alteração, Código 26 de modo a evitar condições de corrida entre tarefas. Assim, procedure **Push** foi substituído por **entry Push**. Uma **entry** é um procedimento com características especiais, tem uma condição de execução associada definida por uma variável do tipo booleana. Uma **entry** só é executada se a condição de execução for verdadeira, ou seja, a tarefa que a evoca, acede ou espera mediante o estado da condição de execução. Desta forma é garantido que somente uma tarefa de cada vez acede aos dados, no entanto, no perfil Ravenscar só é permitida um entry por objecto protegido.

```
001     -- SPARK 2014
002     pragma SPARK_Mode;
003     pragma Profile (Ravenscar);
004     pragma Partition_Elaboration_Policy (Sequential);
005
006
007     package Stack1
008     is
009
010         Stack_Size : constant := 5;
011         type Pointer_Stack is range 0 .. Stack_Size;
012         subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
013         type Vector is array (Index_Stack) of Integer;
014
015      protected type StackPC
016      is
017            function Empty return Boolean
018            ;
019            function Full return Boolean
020            ;
021            entry Push (I : in Integer)
022            ;
023            procedure Pop (I : out Integer)
024            ;
025
026      private
027            Size_of_Stack : Pointer_Stack := 0;
028            Data_in_Stack : Vector := Vector'(Index_Stack => 0);
029            Not_Full      : Boolean := True;
030         end StackPC;
031
032
033      StackTask : StackPC;
034
035      task PushStack
036            with
037                Global => (In_Out => Stacktask),
038            Depends  => (PushStack => PushStack,
039                  StackTask => StackTask)
040           ;
041         task PopStack
042            with
043                Global => (In_Out => Stacktask),
044            Depends  => (PopStack  => PopStack,
045                  StackTask => StackTask)
046         ;
```

```
047
048
049     end Stack1;

001     -- SPARK 2014
002     pragma SPARK_Mode;
003     pragma Profile (Ravenscar);
004     pragma Partition_Elaboration_Policy (Sequential);
005
006     with Gnat.Io; use Gnat.Io;
007     with PrintPopPush;
008
009     package body Stack1
010     is
011
012         protected body StackPC
013         is
014     function Full return Boolean is (Size_of_Stack = Stack_Size);
015
016     function Empty return Boolean is (Size_of_Stack = 0);
017
018             entry Push (I : in Integer) when Not_Full
019             is
020             begin
021                 Not_Full := false;
022                 if not Full then
023                    Size_of_Stack := Size_of_Stack + 1;
024                    Data_in_stack (Size_of_stack) := I;
025                    PrintPopPush ("PUSH", "PushStack - Valor: " & Integer'Image
(Data_in_stack (Size_of_stack)) & " <><><><><>");
026                    pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so'
existe para efeitos de teste");
027                 end if;
028                 Not_Full := True;
029             end Push;
030
031             procedure Pop (I : out Integer)
032             is
033             begin
034         I := 0;
035                 if not Empty then
036                    I := Data_in_stack (Size_of_stack);
037                    Size_of_Stack := Size_of_Stack - 1;
038                    PrintPopPush ("POP", "PopStack - Valor: " & Integer'Image (I)
& " ----------------------------");
039                    pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so'
existe para efeitos de teste");
040                    PrintPopPush ("SIZE", "PopStack - SizeStack: " & Integer'Image
(Integer (Size_of_Stack)) & "");
041                    pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so'
existe para efeitos de teste");
042                    Not_Full := True;
043                 else
044                    Not_Full := true;
045                 end if;
046             end Pop;
047
048         end StackPC;
049
050         task body PushStack
051      is
052             nr     : Integer := 0;
053     T_Full : Boolean := False;
```

```
054         begin
055            loop
056                if nr < Integer'Last then
057                   nr := nr + 1;
058                else
059                   nr := 1;
060                end if;
061                StackTask.Push (nr);
062            end loop;
063         end PushStack;
064
065         task body PopStack is
066            nr    : Integer := 0;
067         begin
068     loop
069                StackTask.Pop (nr);
070            end loop;
071         end PopStack;
072
073     end Stack1;
```

Código 26 – Stack concorrente com entry Push

```
gnatprove    -PD:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese1\stack_prod_cons.gpr    --
level=0 --ide-progress-bar
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
stack1.ads:15:20: info: "Stack1" requires body ("StackPC" requires completion)
stack1.ads:21:15: high: potentially blocking operation in protected operation "Push"
stack1.ads:23:19: high: potentially blocking operation in protected operation "Pop"
stack1.ads:35:10:  info:  "Stack1"  requires  body  (type  derived  from  "PushStack"  requires
completion)
stack1.ads:41:10:  info:  "Stack1"  requires  body  (type  derived  from  "PopStack"  requires
completion)
Summary                                          logged                                        in
D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese1\obj\gnatprove\gnatprove.out
 [2017-06-15 23:59:22] process terminated successfully, elapsed time: 03.79s
```

Figura 18 – Resultado da verificação de Código 26

Com as alterações efetuadas os erros que permanecem são os provocados pelo programa PrintPopPush, que, como referido anteriormente só tem o propósito de teste, devendo ser retirado.

No código seguinte tentou-se implementar o pragma GNAT_Extended_Ravenscar, que é ligeiramente menos restritivo que o Ravenscar, e que segundo a documentação do SPARK 2014 permite a existência de mais do que uma entry. De acordo com o SPARK 2014 User's Guide no capítulo 5.9 só é necessário alterar o pragma profile:

> "To use the GNAT Extended Ravenscar profile simply replace `Ravenscar` with `GNAT_Extended_Ravenscar` in the pragma `Profile` …".

No entanto esta alteração não funcionou. Foi também acrescentado prama Restrictions e alterado o valor de Max_Protected_entries para 2 mas o resultado manteve-se como se pode observar em Figura 18.

```
001     -- SPARK 2014
002     pragma SPARK_Mode;
003     pragma Profile (GNAT_Extended_Ravenscar);
004     pragma Partition_Elaboration_Policy (Sequential);
005  pragma Restrictions(Max_Protected_Entries => 2);
006
007
008
009  package Stack2
010     is
011
012         Stack_Size : constant := 5;
013         type Pointer_Stack is range 0 .. Stack_Size;
014         subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
015      type Vector is array (Index_Stack) of Integer;
016      Size_ini   : Integer := 0;
017
018
019
020         protected type StackPC is
021            function Empty return Boolean
022            ;
023            function Full return Boolean
024            ;
025            entry Push (I : in Integer)
026            ;
027            entry Pop (I : out Integer)
028            ;
029         private
030            Size_of_Stack : Pointer_Stack := 0;
031            Data_in_Stack : Vector := Vector'(Index_Stack => 0);
032            Not_Full      : Boolean := True;
033     Not_Empty     : Boolean := False;
034      end StackPC;
035
036
037         StackTask : StackPC;
038
039         task PushStack
040            with
041                Global => (In_Out => Stacktask),
042            Depends  => (PushStack => PushStack,
043                  StackTask => StackTask)
044           ;
045         task PopStack
046            with
047                Global => (In_Out => Stacktask),
048            Depends  => (PopStack  => PopStack,
049           StackTask => StackTask)
050         ;
051
052     end Stack2;
```

```
001     -- SPARK 2014
002     pragma SPARK_Mode;
003     pragma Profile (GNAT_Extended_Ravenscar);
004     pragma Partition_Elaboration_Policy (Sequential);
005  pragma Restrictions (Max_Protected_Entries => 2);
006
007     with Gnat.Io; use Gnat.Io;
008  with PrintPopPush;
009
010     package body Stack2
```

```
011     is
012         protected body StackPC
013         is
014     function Full return Boolean is (Size_of_Stack = Stack_Size);
015
016     function Empty return Boolean is (Size_of_Stack = 0);
017
018             entry Push (I : in Integer) when Not_Full
019             is
020             begin
021                 Not_Full := True;
022                 if not Full then
023                    Size_of_Stack := Size_of_Stack + 1;
024                    Data_in_stack (Size_of_stack) := I;
025                    PrintPopPush ("PUSH", "PushStack - Valor: " & Integer'Image
(Data_in_stack (Size_of_stack)) & " <><><><><>");
026                    pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so'
existe para efeitos de teste");
027                 end if;
028                 Not_Full := True;
029             end Push;
030
031             entry Pop (I : out Integer) when Not_Empty
032             is
033     begin
034                 Not_Empty := False;
035                 if not Empty then
036                    I := Data_in_stack (Size_of_stack);
037                    Size_of_Stack := Size_of_Stack - 1;
038                    PrintPopPush ("POP", "PopStack - Valor: " & Integer'Image (I)
& " ----------------------------");
039                    pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so'
existe para efeitos de teste");
040                    PrintPopPush ("SIZE", "PopStack - SizeStack: " & Integer'Image
(Integer (Size_of_Stack)) & "");
041                    pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so'
existe para efeitos de teste");
042                 end if;
043                 Not_Empty := True;
044             end Pop;
045
046         end StackPC;
047
048      task body PushStack
049         is
050     nr      : Integer := 0;
051             T_Full : Boolean := False;
052         begin
053            loop
054                T_Full := StackTask.Full;
055                if  not T_Full then
056                   if nr < Integer'Last then
057                      nr := nr + 1;
058                   else
059                      nr := 1;
060                   end if;
061                   StackTask.Push (nr);
062                end if;
063            end loop;
064         end PushStack;
065
066         task body PopStack is
067            nr      : Integer := 0;
```

```
068     T_Empty : Boolean := False;
069         begin
070            loop
071                T_Empty := StackTask.Empty;
072                if not T_Empty then
073                   StackTask.Pop (nr);
074                end if;
075
076            end loop;
077         end PopStack;
078
079     end Stack2;
```

Código 27 – Stack concorrente com entry Push e entry Pop

```
gnatprove     -PD:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese2\stack_prod_cons.gpr     --
level=0 --ide-progress-bar
Phase 1 of 2: generation of Global contracts ...
stack2.ads:21:09: violation of restriction "Max_Protected_Entries = 1" at stack2.adb:3
gnatprove: error during generation of Global contracts
[2017-08-16 00:22:25] process exited with status 1, elapsed time: 00.62s
```

Figura 19 – Resultado da verificação de Código 27

6 Evolução do SPARK

A evolução do SPARK tem acompanhado a evolução da linguagem Ada, sendo o SPARKA2014 a versão mais recente baseada no Ada2012. A nova versão a sofreu grande alteração face à versão SPARK2005, aumentou o subconjunto do Ada utilizado tornando-a mais flexível e introduziu uma nova ferramenta de prova e verificação que substitui as anteriores. Com esta nova versão surge também uma ferramenta de teste, que gera de forma automática código para testar o código desenvolvido.

6.1 Diferenças entre SPARK2005 e SPARK2014

Na tabela seguinte apresentam-se as algumas diferenças entre as versões 2005 e 2014 da linguagem SPARK.

Tabela 5 – Diferenças entre SPARK2005 e SPARK2014

	SPARK2005	**SPARK2014**
Ferramenta de Verificação	• Examiner • Simplifier • Proof Checker • POGS	• GNATProve
Ferramenta de Testes		• GNATtest
Anotações/Contratos	Em forma de comentários Ada	Adoção da nova característica do Ada 2012 "**aspects**" que permite a escrita de contratos de forma mais legível
Proof Functions	Funções usadas nas anotações para verificação e prova	

Ghost Funtions		Equivalente à Proof Function em SPARK2005. Não faz parte do código compilado
Expression Functions		Funções que não requerem corpo da função, retorna a expressão, exemplo: `function Incr (X : Integer) return Integer is (X + 1);`
Contract Cases		Pode ser usado em substituição de postconditions complexas facilitando a leitura/compreensão do contrato
Predicates		Usado para indicar propriedades invariantes de tipos. Por exemplo, o tipo de dado Count não pode ser igual a 10 ou 100, então define-se da seguinte forma: `type Count is new Natural with Predicate => Count /= 10 and Count /= 100;`
Loop_Variants		Asserção usada para prova de terminação de ciclos.
Concorrência	• Ravenscar profile	• Ravenscar profile • GNAT_Extended_Ravenscar, baseado no Ravenscar mas menos restritivo

Das diferenças referidas na tabela anterior, a ferramenta de prova, GNATprove, e a forma como os anotações/contratos passaram a ser escritos serão as mais significativas. A adoção da plataforma Why3 para prova do código permite uma maior versatilidade à ferramenta, na medida em que pode utilizar os vários SMTs existentes, desde que disponíveis na plataforma. O resultado produzido pelo GNATprove durante a verificação do código é mais claro que o resultante nas versões anteriores, mas, em alguns casos as mensagens de erro não são suficientemente claras. Esta ferramenta, embora utilize os contratos para verificar o código, na ausência destes tem capacidade para geração de contratos com vista a proceder à análise do programa. No entanto, deve-se evitar esta prática pois pode conduzir a resultados menos corretos. As anotações utilizadas nesta versão, adotadas do Ada 2012, são mais explícitas tornando o código mais legível, logo, permite uma melhor compreensão do que é pretendido. Apesar desta melhoria, a elaboração de contratos não deixa de ser uma tarefa complexa e que continua a ter uma componente de tempo significativa no desenvolvimento de software. Elaborar um contrato pode ser mais difícil que desenvolver o algoritmo que se pretende provar como correto.

As anotações utilizadas nesta versão são mais explícitas tornando o código mais legível, logo, permite uma melhor compreensão do que é pretendido. Apesar desta melhoria, a elaboração de contratos ainda é uma tarefa complexa e que continua a ter uma componente de tempo significativa no desenvolvimento de software.

A nova ferramenta GNATtest permite criar e manter uma infra-estrutura de teste completa para os projetos desenvolvidos. Permite, que à medida que se acrescentam novos subprogramas sejam criadas novas unidades de teste.Cada subprograma visível deve ter pelo menos um teste unitário correspondente.

O código Ghost é uma característica bastante interessante, na medida em que se pode desenvolver funções ou procedimentos para elaborar contratos, mas não é incluído na versão executável do programa. Em termos de verificação o GNATProve atua se fosse código normal e simultaneamente verifica se este não interfere no comportamento do programa. Esta funcionalidade é bastante útil e pode utilizada em várias situações. Por exemplo:

- Funções Ghost – muito uteis para a escrita de contratos, e do ponto de vista do programador, são uma mais-valia na medida em que não é necessário escrever o corpo da função bastando definir o valor de retorno;
- Variáveis Ghost – usadas para guardar valores que serão utilizados em asserções;
- Tipos Ghost – existem somente para definição de variáveis Ghost não podendo ser usados para fim.

A criação do perfil GNAT_Extended_Ravenscar, menos restritivo que o Ravenscar, veio permitir a utilização de um maior subconjunto da linguagem Ada no que à concorrência diz respeito. Este novo perfil foi criado essencialmente para sistemas de tempo real e embebidos. Também é válida a utilização deste perfil em sistemas que não estejam sujeitos certificação formal. Segundo o SPARK 2014 User's Guide, para utilizar este perfil basta substituir Ravenscar por GNAT_Extended_Ravenscar. Em Código 27 foi feita esta alteração, mas o resultado obtido é o que consta da Figura 19. Poderá ser necessário alterar ou adicionar alguma configuração, mas não foi encontrada qualquer informação nos manuais, quer no User's Guide quer no Reference Manual.

7 Conclusões

Nesta dissertação estudou-se o desenvolvimento e a usabilidade de algoritmos simples corretos pela verificação formal das suas propriedades, para isso recorreu-se à linguagem SPARK, versão 2014, e em alguns casos procedeu-se à comparação com a versão anterior. Os programas desenvolvidos utilizam estruturas de dados simples, de uso comum nas aplicações da área do software de alta fiabilidade e integridade. Durante o desenvolvimento foram implementados contratos com o intuito de demonstrar a correcção do código.

As linguagens de programação que usam técnicas de verificação formal de software, como o SPARK, ainda estão restritas a domínios muito específicos. A informação disponível sobre SPARK2014 em particular não é muito abundante, e a que existe é, de forma geral, muito técnica e dirigida a um público muto específico. Apesar disso, os documentos lidos e o código produzido no decurso do desenvolvimento deste projeto, permitem alguns comentários quanto à utilidade/usabilidade do SPARK.

Ainda é uma linguagem essencialmente dirigida ao desenvolvimento de sistemas críticos, que requerem alta integridade e fiabilidade. Embora a estrutura da linguagem seja diferente daquelas mais usadas, depois de se ler a documentação e analisar alguns exemplos, o desenvolvimento de código faz-se com relativa rapidez. No entanto, conhecer a linguagem Ada, embora não seja requisito essencial, é aconselhável e uma mais-valia para o desenvolvimento em SPARK.

A introdução de contratos formais na própria linguagem, adotados do Ada 2012, e a utilização de uma plataforma de verificação aberta, Why3, por parte da ferramenta de prova GNATProve trouxe uma flexibilidade na análise e verificação de código muito grande. O facto de o Why3 permitir inúmeros SMT e o GNATProve poder usufruir da sua utilização, proporciona uma maior capacidade de prova automática à ferramenta. A opção por esta solução de prova e verificação é um grande avanço face às versões anteriores de SPARK, se não mesmo o maior.

Nem sempre o código produzido tem os elementos necessários e/ou suficientes, para que o GNATProve possa proceder à verificação e prove a ausência de erros. Com a versão SPARK21014 foi introduzida uma nova funcionalidade, Ghost, que permite desenvolver funções e criar variáveis para elaborar pré-condições e pós-condições com a finalidade de provar que programa está isento de erros, não sendo este código incluído no executável final.

No entanto, a componente mais difícil e morosa continua a ser a elaboração de contratos. Embora a escrita dos mesmos esteja mais facilitada, podendo ser usadas expressões condicionais (if..else, case) esta é uma tarefa que requer muito treino, só desta forma se conseguirá agilidade necessária para definir e estabelecer contratos. Nos exemplos apresentados, o tempo despendido na formulação das anotações foi sempre superior ao da escrita do código de programas. A pouca experiência neste tipo de desenvolvimento resultou na situação descrita. Poderá parecer um pouco exagerado, visto tratarem-se de estruturas de dados simples, mas a formulação de um contrato que satisfaça a condição de verificação pode ser complexa, mesmo quando se trata da eliminação de um erro que pode parecer trivial.

Por outro lado, o tempo despendido no estabelecimento de contratos que satisfazem as especificações, em ambiente real, será recuperado na fase de testes e implementação devido ao elevado grau de certeza de que o aplicativo está correto.

Como já foi referido, os maiores obstáculos encontrados neste projeto foi o desenvolvimento dos contratos. O exemplo que resolve o problema de N-Queens de forma iterativa ilustra bem a dificuldade em escrever contratos para provar formalmente a sua correção. Embora o código esteja correto, após várias hipóteses de contratos que não resultaram, foi implementado um pragma Annotate para ultrapassar as mensagens de erro produzidas pelo GNATProve. Esta é uma solução que só deve ser utilizada caso se tenha a certeza que o erro, identificado pela ferramenta de verificação, não acontece.

Outra dificuldade sentida ao longo deste trabalho prende-se com algumas mensagens de erro ou informativas produzidas pelo GNATProve. Para um programador com pouca experiência nesta linguagem, nem sempre estas são percetíveis. Sendo o número de utilizadores, de SPARK, pequeno quando comparado com os números de utilizadores de outras linguagens, torna-se difícil encontrar respostas pesquisando na Internet que ajudem a perceber qual o problema.

Para quem se inicia numa linguagem de programação, os manuais desta são as principais ferramentas de aprendizagem. No caso do SPARK, existem algumas informações menos precisas que podem induzir em erro um iniciante. Por exemplo, a utilização do pragma GNAT_Extendeded_Ravenscar permite a utilização de mais do que uma entry por objeto protegido. No entanto, em Código 27, para efeitos de demonstração foi usado este pragma produzindo o resultado constante na Figura 19. Poderá ser necessário proceder a mais algumas alterações no código ou configuração, mas nos manuais nada é indicado e a mensagem de erro não elucidativa.

Em relação ao futuro, percebe-se que esta é uma linguagem em evolução, acrescentando novas funcionalidades sem perder de vista o seu objetivo principal, a criação de software formalmente

correto. Nesta versão o subconjunto da linguagem Ada que compõe o SPARK foi alargado, foram incorporadas algumas novas características do Ada 2012, caso dos **aspects**. O GNAT_Extended_Ravenscar um perfil menos restritivo que o Ravenscar e que faz parte do SPARK, poderá vir a integrar no futuro a linguagem Ada. Percebe-se que as pessoas que se dedicam ao desenvolvimento da linguagem têm feito um esforço para incorporar cada vez mais funcionalidades do Ada no SPARK. Este percurso, poderá no futuro conduzir à reunificação do SPARK com o Ada. Se tal junção acontecer, o potencial do Ada para desenvolvimento de aplicações mais comerciais com a capacidade de verificação formal de software do SPARK, será o emergir de uma linguagem bastante poderosa.

Se o SPARK ou uma possível nova linguagem resultante da junção Ada com Spark, se expandir para outras áreas de negócio, e as empresas perceberem a mais-valia que é o desenvolvimento e/ou a aquisição de software que foi objeto de verificação formal, poderá vir a ter um papel de relevo no panorama do software. Mas para isso acontecer é necessário que a linguagem se abra ao mundo. O facto de a empresa proprietária da linguagem disponibilizar um ambiente de desenvolvimento gratuito, embora não sendo tão evoluído como os de outras linguagens, permite às escolas dar a conhecer às novas gerações o potencial desta linguagem. Com a divulgação desta linguagem, das suas ferramentas de verificação e mostrar a vantagem do uso destas ferramentas, o número de utilizadores não dedicados ao desenvolvimento de software crítico tenderá a aumentar.

Referências

[1] K. Weiss, “Software Sector Is Set for Strong Growth in 2017 | Morgan Stanley,” 2017. [Online]. Available: https://www.morganstanley.com/ideas/software-sector-growth. [Accessed: 11-Jun-2017].

[2] IDC, “IDC Forecasts Worldwide Spending on Digital Transformation Technologies Will Surpass $2 Trillion in 2019,” 2016. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS42327517. [Accessed: 10-Jun-2017].

[3] “Software Glitch Causes F-35 to Incorrectly Detect Targets in Formation - Defensetech.” [Online]. Available: https://www.defensetech.org/2015/03/24/software-glitch-causes-f-35-to-incorrectly-detect-targets-in-formation/. [Accessed: 15-Jun-2017].

[4] P. Amey, “Can Also Be Cheaper Correctness by Construction : Better Can Also Be Cheaper,” *Defense*, no. March, 2002.

[5] “SPARK,” 22-Jan-2016. [Online]. Available: http://intelligent-systems.altran.com/technologies/software-engineering/spark.html. [Accessed: 22-Dec-2015].

[6] “About SPARK 2014.” [Online]. Available: http://www.spark-2014.org/about/. [Accessed: 08-Jan-2016].

[7] “Why3.” [Online]. Available: http://why3.lri.fr/. [Accessed: 12-Jan-2016].

[8] C. A. R. Hoare, “An Axiomatic Basis for Computer Programming,” *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[9] R. W. Floyd, “FloydMeaning.pdf,” in *Mathematical Aspects of Computer Science*, J. T. Schwartz, Ed. Proceedings of the American Mathematical Society Symposia on Applied Mathematics, 1967, pp. 19–32.

[10] “What is DO-178C? | DO-178B/DO-178C Avionics | GNAT Pro Safety-Critical | AdaCore.” [Online]. Available: http://www.adacore.com/gnatpro-safety-critical/avionics/do178c/. [Accessed: 19-Feb-2016].

[11] “History’s Worst Software Bugs.” [Online]. Available: http://archive.wired.com/software/coolapps/news/2005/11/69355?currentPage=all. [Accessed: 27-Jan-2016].

[12] “Ariane 501 - Presentation of Inquiry Board report / Press Releases / For Media / ESA.” [Online]. Available: http://www.esa.int/For_Media/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report. [Accessed: 27-Jan-2016].

[13] “GAO Report: Patriot Missile Defense-- Software Problem Led to System Failure at Dhahran, Saudi Arabia.” [Online]. Available: http://fas.org/spp/starwars/gao/im92026.htm. [Accessed: 27-Jan-2016].

[14] J. Barnes, *High Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[15] R. Chapman and F. Schanda, “Are we there yet? 20 years of industrial theorem proving with spark,” in *Interactive Theorem Proving*, no. March 1987, Springer, 2014, pp. 17–26.

[16] “MIL-STD-1815A NOT 1 Ada Programming Language (Ansi/Mil Std 1815a-1983) (No S/S Document).” [Online]. Available: http://webstore.ansi.org/RecordDetail.aspx?sku=MIL-STD-1815A+NOT+1. [Accessed: 18-Feb-2016].

[17] “ISO 8652:1987 - Programming languages -- Ada.” [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=16028. [Accessed: 18-Feb-2016].

[18] R. Chapman, “Recent results with Correctness by Construction and SPARK Rod Chapman Praxis High Integrity Systems,” *Praxis (Bern. 1994).*, 2005.

[19] M. Croxford and R. Chapman, “Correctness by Construction: A Manifesto for High-Integrity Software,” *J. Def. Softw. Eng.*, no. Spivey 1992, pp. 5–8, 2005.

[20] B. Meyer, “Applying ‘Design by Contract,’” *Computer (Long. Beach. Calif).*, vol. 25, no. 10, pp. 40–51, 1992.

[21] “The lessons of Ariane.” [Online]. Available: http://www.irisa.fr/pampa/EPEE/Ariane5.html. [Accessed: 27-Jan-2016].

[22] M. J. Frade and J. S. Pinto, “Verification conditions for source-level imperative programs,” *Comput. Sci. Rev.*, vol. 5, no. 3, pp. 252–277, 2011.

[23] T. Weber, “Satisfiability Modulo Theories,” *Handb. Satisf.*, vol. 185, pp. 825–885, 2004.

[24] R. Chapman, “Industrial experience with SPARK,” *ACM SIGAda Ada Lett.*, vol. XX, no. 4, pp. 64–68, 2000.

[25] S. King, J. Hammond, R. Chapman, and A. Pryor, “FM’99 --- Formal Methods: World Congress on Formal Methods in the Development of Computing Systems Toulouse, France, September 20--24, 1999 Proceedings, Volume II,” J. M. Wing, J. Woodcock, and J. Davies, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 1527–1545.

[26] M. Croxfore, J. Sutton, P. C. Systems, and L. A. S. Company, “SPARK - A successful contribution to the Lockheed C130-J Hercules II,” *Praxis (Bern. 1994).*

[27] “Flight Software for Lunar IceCube Satellite | Customers | AdaCore.” [Online]. Available: http://www.adacore.com/customers/flight-software-for-lunar-icecube-satellite/. [Accessed: 02-Jan-2016].

[28] “Introduction.” [Online]. Available: http://www.ada-auth.org/standards/12rat/html/Rat12-1.html. [Accessed: 20-Dec-2015].

[29] “SPARK 2014 User’s Guide — SPARK 2014 User’s Guide 17.0w documentation.” [Online]. Available: http://docs.adacore.com/spark2014-docs/html/ug/index.html. [Accessed: 19-Feb-2016].

[30] “Aunit | Tools | Libre.” [Online]. Available: http://libre.adacore.com/tools/aunit/. [Accessed: 19-Feb-2016].

[31] “AUnit Cookbook.” [Online]. Available: http://docs.adacore.com/aunit-docs/aunit.html. [Accessed: 19-Feb-2016].

[32] Why Development Team, “Why: A Software Verification Platform.” .

[33] SPARK Team, “SPARK Examiner The SPARK Ravenscar Profile,” 2004.

Anexos

Anexo I

```
          ***************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ***************************************************


                       DATE : 11-JAN-2016 22:35:13.44

                       procedure dissertacao.Troca

For path(s) from start to run-time check associated with statement of line 8:

procedure_troca_1.
H1:    x >= float__first .
H2:    x <= float__last .
H3:    y >= float__first .
H4:    y <= float__last .
       ->
C1:    x >= float__first .
C2:    x <= float__last .


For path(s) from start to run-time check associated with statement of line 9:

procedure_troca_2.
H1:    x >= float__first .
H2:    x <= float__last .
H3:    y >= float__first .
H4:    y <= float__last .
H5:    x >= float__first .
H6:    x <= float__last .
       ->
C1:    y >= float__first .
C2:    y <= float__last .


For path(s) from start to run-time check associated with statement of line 10:

procedure_troca_3.
H1:    x >= float__first .
H2:    x <= float__last .
H3:    y >= float__first .
H4:    y <= float__last .
H5:    x >= float__first .
H6:    x <= float__last .
H7:    y >= float__first .
H8:    y <= float__last .
H9:    x >= float__first .
H10:   x <= float__last .
       ->
C1:    x >= float__first .
C2:    x <= float__last .


For path(s) from start to finish:

procedure_troca_4.
H1:    x >= float__first .
H2:    x <= float__last .
H3:    y >= float__first .
H4:    y <= float__last .
H5:    x >= float__first .
H6:    x <= float__last .
H7:    y >= float__first .
H8:    y <= float__last .
H9:    x >= float__first .
H10:   x <= float__last .
H11:   x >= float__first .
H12:   x <= float__last .
       ->
C1:    y = y .
C2:    x = x .
```

Conteúdo do ficheiro troca.vcg

Anexo II

```
******************************************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
******************************************************************************


CREATED 11-JAN-2016, 22:40:20  SIMPLIFIED 11-JAN-2016, 22:35:13

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

procedure dissertacao.Troca




For path(s) from start to run-time check associated with statement of line 8:

procedure_troca_1.
*** true .          /* all conclusions proved */


For path(s) from start to run-time check associated with statement of line 9:

procedure_troca_2.
*** true .          /* all conclusions proved */


For path(s) from start to run-time check associated with statement of line 10:

procedure_troca_3.
*** true .          /* all conclusions proved */


For path(s) from start to finish:

procedure_troca_4.
*** true .          /* all conclusions proved */
```

Conteúdo do ficheiro troca.siv

Anexo III

```
           ******************************************************
                       Semantic Analysis of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ******************************************************


                        DATE : 16-JUN-2016 17:31:58.29

                          procedure Stack2005.Push



For path(s) from start to run-time check associated with statement of line 9:

procedure_push_1.
H1:    not (full(s)) .
H2:    fld_size_of_stack(s) >= pointer_stack__first .
H3:    fld_size_of_stack(s) <= pointer_stack__last .
H4:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H5:    i >= integer__first .
H6:    i <= integer__last .
        ->
C1:    fld_size_of_stack(s) + 1 >= pointer_stack__first .
C2:    fld_size_of_stack(s) + 1 <= pointer_stack__last .


For path(s) from start to run-time check associated with statement of line 10:

procedure_push_2.
H1:    not (full(s)) .
H2:    fld_size_of_stack(s) >= pointer_stack__first .
H3:    fld_size_of_stack(s) <= pointer_stack__last .
H4:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H5:    i >= integer__first .
H6:    i <= integer__last .
H7:    fld_size_of_stack(s) + 1 >= pointer_stack__first .
H8:    fld_size_of_stack(s) + 1 <= pointer_stack__last .
        ->
C1:    i >= integer__first .
C2:    i <= integer__last .
C3:    fld_size_of_stack(upf_size_of_stack(s, fld_size_of_stack(
           s) + 1)) >= index_stack__first .
C4:    fld_size_of_stack(upf_size_of_stack(s, fld_size_of_stack(
           s) + 1)) <= index_stack__last .


For path(s) from start to finish:

procedure_push_3.
H1:    not (full(s)) .
H2:    fld_size_of_stack(s) >= pointer_stack__first .
H3:    fld_size_of_stack(s) <= pointer_stack__last .
```

```
H4:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H5:    i >= integer__first .
H6:    i <= integer__last .
H7:    fld_size_of_stack(s) + 1 >= pointer_stack__first .
H8:    fld_size_of_stack(s) + 1 <= pointer_stack__last .
H9:    i >= integer__first .
H10:   i <= integer__last .
H11:   fld_size_of_stack(upf_size_of_stack(s, fld_size_of_stack(
           s) + 1)) >= index_stack__first .
H12:   fld_size_of_stack(upf_size_of_stack(s, fld_size_of_stack(
           s) + 1)) <= index_stack__last .
        ->
C1:    not (empty(upf_data_in_stack(upf_size_of_stack(s,
           fld_size_of_stack(s) + 1), update(fld_data_in_stack(
           upf_size_of_stack(s, fld_size_of_stack(s) + 1)), [
           fld_size_of_stack(upf_size_of_stack(s, fld_size_of_stack(
           s) + 1))], i)))) .



           ****************************************************
                       Semantic Analysis of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ****************************************************


                        DATE : 16-JUN-2016 17:31:58.29

                           procedure Stack2005.Pop



For path(s) from start to run-time check associated with statement of line 18:

procedure_pop_1.
H1:    not (empty(s)) .
H2:    fld_size_of_stack(s) >= pointer_stack__first .
H3:    fld_size_of_stack(s) <= pointer_stack__last .
H4:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
        ->
C1:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
C2:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .
C3:    fld_size_of_stack(s) >= index_stack__first .
C4:    fld_size_of_stack(s) <= index_stack__last .


For path(s) from start to run-time check associated with statement of line 19:

procedure_pop_2.
H1:    not (empty(s)) .
H2:    fld_size_of_stack(s) >= pointer_stack__first .
H3:    fld_size_of_stack(s) <= pointer_stack__last .
H4:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
```

```
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H5:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
H6:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .
H7:    fld_size_of_stack(s) >= index_stack__first .
H8:    fld_size_of_stack(s) <= index_stack__last .
        ->
C1:    fld_size_of_stack(s) - 1 >= pointer_stack__first .
C2:    fld_size_of_stack(s) - 1 <= pointer_stack__last .


For path(s) from start to finish:

procedure_pop_3.
H1:    not (empty(s)) .
H2:    fld_size_of_stack(s) >= pointer_stack__first .
H3:    fld_size_of_stack(s) <= pointer_stack__last .
H4:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H5:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
H6:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .
H7:    fld_size_of_stack(s) >= index_stack__first .
H8:    fld_size_of_stack(s) <= index_stack__last .
H9:    fld_size_of_stack(s) - 1 >= pointer_stack__first .
H10:   fld_size_of_stack(s) - 1 <= pointer_stack__last .
        ->
C1:    not (full(upf_size_of_stack(s, fld_size_of_stack(s) - 1))) .


           ****************************************************
                       Semantic Analysis of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ****************************************************


                        DATE : 16-JUN-2016 17:31:58.29

                         procedure Stack2005.Clear




For path(s) from start to run-time check associated with statement of line 26:

procedure_clear_1.
H1:    fld_size_of_stack(s) >= pointer_stack__first .
H2:    fld_size_of_stack(s) <= pointer_stack__last .
H3:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
        ->
C1:    0 >= pointer_stack__first .
C2:    0 <= pointer_stack__last .
```

```
For path(s) from start to finish:

procedure_clear_2.
H1:    fld_size_of_stack(s) >= pointer_stack__first .
H2:    fld_size_of_stack(s) <= pointer_stack__last .
H3:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H4:    0 >= pointer_stack__first .
H5:    0 <= pointer_stack__last .
        ->
C1:    empty(upf_size_of_stack(s, 0)) .


           ******************************************************
                       Semantic Analysis of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ******************************************************


                        DATE : 16-JUN-2016 17:31:58.29

                           function Stack2005.Top



For path(s) from start to run-time check associated with statement of line 37:

function_top_1.
H1:    fld_size_of_stack(s) >= pointer_stack__first .
H2:    fld_size_of_stack(s) <= pointer_stack__last .
H3:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H4:    fld_size_of_stack(s) > 0 .
        ->
C1:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
C2:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .
C3:    fld_size_of_stack(s) >= index_stack__first .
C4:    fld_size_of_stack(s) <= index_stack__last .


For path(s) from start to finish:

function_top_2.
H1:    fld_size_of_stack(s) >= pointer_stack__first .
H2:    fld_size_of_stack(s) <= pointer_stack__last .
H3:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H4:    fld_size_of_stack(s) > 0 .
H5:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
H6:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .
```

```
H7:    fld_size_of_stack(s) >= index_stack__first .
H8:    fld_size_of_stack(s) <= index_stack__last .
H9:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
H10:   element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .
        ->
C1:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) = element(
           fld_data_in_stack(s), [fld_size_of_stack(s)]) .
C2:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >=
           integer__first .
C3:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <=
           integer__last .


function_top_3.
H1:    fld_size_of_stack(s) >= pointer_stack__first .
H2:    fld_size_of_stack(s) <= pointer_stack__last .
H3:    for_all(i___1: pointer_stack, ((i___1 >= index_stack__first) and (
           i___1 <= index_stack__last)) -> ((element(fld_data_in_stack(
           s), [i___1]) >= integer__first) and (element(
           fld_data_in_stack(s), [i___1]) <= integer__last))) .
H4:    not (fld_size_of_stack(s) > 0) .
H5:    0 >= integer__first .
H6:    0 <= integer__last .
        ->
C1:    0 = element(fld_data_in_stack(s), [fld_size_of_stack(
           s)]) .
C2:    0 >= integer__first .
C3:    0 <= integer__last .


For checks of refinement integrity:

function_top_4.
*** true .          /* trivially true VC removed by Examiner */


function_top_5.
*** true .          /* trivially true VC removed by Examiner */



           ******************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ******************************************************


                         DATE : 16-JUN-2016 17:31:58.29

                           function Stack2005.Empty




For path(s) from start to finish:

function_empty_1.
*** true .          /* trivially true VC removed by Examiner */
```

```
           ******************************************************
                       Semantic Analysis of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
           ******************************************************


                        DATE : 16-JUN-2016 17:31:58.29

                          function Stack2005.Full



For path(s) from start to finish:

function_full_1.
*** true .          /* trivially true VC removed by Examiner */
```

Conteúdo dos ficheiros Push.vcg, Pop.vcg, Clear.vcg, Top.vcg, Empty.vcg e Full.vcg

Anexo IV

```
**************************************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
**************************************************************************


CREATED 16-JUN-2016, 18:05:50  SIMPLIFIED 16-JUN-2016, 18:06:14

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

procedure Stack2005.Push




For path(s) from start to run-time check associated with statement of line 7:

procedure_push_1.
H1:    not full(s) .
H2:    fld_size_of_stack(s) >= 0 .
H3:    fld_size_of_stack(s) <= 10 .
H4:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 10 -> integer__first <=
          element(fld_data_in_stack(s), [i___1]) and element(fld_data_in_stack(
          s), [i___1]) <= integer__last) .
H5:    i >= integer__first .
H6:    i <= integer__last .
H7:    integer__size >= 0 .
H8:    integer__first <= integer__last .
H9:    integer__base__first <= integer__base__last .
H10:   integer__base__first <= integer__first .
H11:   integer__base__last >= integer__last .
H12:   stack__size >= 0 .
H13:   pointer_stack__size >= 0 .
H14:   pointer_stack__base__first <= pointer_stack__base__last .
H15:   index_stack__size >= 0 .
H16:   index_stack__base__first <= index_stack__base__last .
H17:   pointer_stack__base__first <= 0 .
H18:   pointer_stack__base__last >= 10 .
H19:   index_stack__base__first <= 1 .
H20:   index_stack__base__last >= 10 .
       ->
C1:    fld_size_of_stack(s) <= 9 .


For path(s) from start to run-time check associated with statement of line 8:

procedure_push_2.
*** true .          /* all conclusions proved */


For path(s) from start to finish:

procedure_push_3.
H1:    not full(s) .
H2:    fld_size_of_stack(s) >= 0 .
H3:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 10 -> integer__first <=
          element(fld_data_in_stack(s), [i___1]) and element(fld_data_in_stack(
          s), [i___1]) <= integer__last) .
```

```
H4:    i >= integer__first .
H5:    i <= integer__last .
H6:    fld_size_of_stack(s) <= 9 .
H7:    integer__size >= 0 .
H8:    integer__first <= integer__last .
H9:    integer__base__first <= integer__base__last .
H10:   integer__base__first <= integer__first .
H11:   integer__base__last >= integer__last .
H12:   stack__size >= 0 .
H13:   pointer_stack__size >= 0 .
H14:   pointer_stack__base__first <= pointer_stack__base__last .
H15:   index_stack__size >= 0 .
H16:   index_stack__base__first <= index_stack__base__last .
H17:   pointer_stack__base__first <= 0 .
H18:   pointer_stack__base__last >= 10 .
H19:   index_stack__base__first <= 1 .
H20:   index_stack__base__last >= 10 .
       ->
C1:    not empty(upf_size_of_stack(upf_data_in_stack(s, update(
          fld_data_in_stack(s), [fld_size_of_stack(s) + 1], i)),
          fld_size_of_stack(s) + 1)) .


******************************************************************************
                      Semantic Analysis of SPARK Text
                              Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
******************************************************************************


CREATED 16-JUN-2016, 18:05:50  SIMPLIFIED 16-JUN-2016, 18:06:14

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

procedure Stack2005.Pop




For path(s) from start to run-time check associated with statement of line 14:

procedure_pop_1.
H1:    not empty(s) .
H2:    fld_size_of_stack(s) >= 0 .
H3:    fld_size_of_stack(s) <= 10 .
H4:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 10 -> integer__first <=
          element(fld_data_in_stack(s), [i___1]) and element(fld_data_in_stack(
          s), [i___1]) <= integer__last) .
H5:    integer__size >= 0 .
H6:    integer__first <= integer__last .
H7:    integer__base__first <= integer__base__last .
H8:    integer__base__first <= integer__first .
H9:    integer__base__last >= integer__last .
H10:   stack__size >= 0 .
H11:   pointer_stack__size >= 0 .
H12:   pointer_stack__base__first <= pointer_stack__base__last .
H13:   index_stack__size >= 0 .
H14:   index_stack__base__first <= index_stack__base__last .
H15:   pointer_stack__base__first <= 0 .
H16:   pointer_stack__base__last >= 10 .
H17:   index_stack__base__first <= 1 .
H18:   index_stack__base__last >= 10 .
```

```
       ->
C1:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >= integer__first .
C2:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <= integer__last .
C3:    fld_size_of_stack(s) >= 1 .


For path(s) from start to run-time check associated with statement of line 15:

procedure_pop_2.
*** true .          /* all conclusions proved */


For path(s) from start to finish:

procedure_pop_3.
H1:    not empty(s) .
H2:    fld_size_of_stack(s) <= 10 .
H3:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 10 -> integer__first <=
          element(fld_data_in_stack(s), [i___1]) and element(fld_data_in_stack(
          s), [i___1]) <= integer__last) .
H4:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) >= integer__first .
H5:    element(fld_data_in_stack(s), [fld_size_of_stack(s)]) <= integer__last .
H6:    fld_size_of_stack(s) >= 1 .
H7:    integer__size >= 0 .
H8:    integer__first <= integer__last .
H9:    integer__base__first <= integer__base__last .
H10:   integer__base__first <= integer__first .
H11:   integer__base__last >= integer__last .
H12:   stack__size >= 0 .
H13:   pointer_stack__size >= 0 .
H14:   pointer_stack__base__first <= pointer_stack__base__last .
H15:   index_stack__size >= 0 .
H16:   index_stack__base__first <= index_stack__base__last .
H17:   pointer_stack__base__first <= 0 .
H18:   pointer_stack__base__last >= 10 .
H19:   index_stack__base__first <= 1 .
H20:   index_stack__base__last >= 10 .
       ->
C1:    not full(upf_size_of_stack(s, fld_size_of_stack(s) - 1)) .


******************************************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
******************************************************************************


CREATED 16-JUN-2016, 18:05:50  SIMPLIFIED 16-JUN-2016, 18:06:14

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

procedure Stack2005.Clear




For path(s) from start to run-time check associated with statement of line 21:

procedure_clear_1.
*** true .          /* all conclusions proved */
```

```
For path(s) from start to finish:

procedure_clear_2.
H1:    fld_size_of_stack(s) >= 0 .
H2:    fld_size_of_stack(s) <= 10 .
H3:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 10 -> integer__first <=
          element(fld_data_in_stack(s), [i___1]) and element(fld_data_in_stack(
          s), [i___1]) <= integer__last) .
H4:    integer__size >= 0 .
H5:    integer__first <= integer__last .
H6:    integer__base__first <= integer__base__last .
H7:    integer__base__first <= integer__first .
H8:    integer__base__last >= integer__last .
H9:    stack__size >= 0 .
H10:   pointer_stack__size >= 0 .
H11:   pointer_stack__base__first <= pointer_stack__base__last .
H12:   index_stack__size >= 0 .
H13:   index_stack__base__first <= index_stack__base__last .
H14:   pointer_stack__base__first <= 0 .
H15:   pointer_stack__base__last >= 10 .
H16:   index_stack__base__first <= 1 .
H17:   index_stack__base__last >= 10 .
       ->
C1:    empty(upf_size_of_stack(s, 0)) .


****************************************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
****************************************************************************


CREATED 16-JUN-2016, 18:05:50  SIMPLIFIED 16-JUN-2016, 18:06:14

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

function Stack2005.Top




For path(s) from start to run-time check associated with statement of line 30:

function_top_1.
*** true .          /* all conclusions proved */


For path(s) from start to finish:

function_top_2.
*** true .          /* all conclusions proved */


function_top_3.
H1:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 10 -> integer__first <=
          element(fld_data_in_stack(s), [i___1]) and element(fld_data_in_stack(
          s), [i___1]) <= integer__last) .
H2:    0 >= integer__first .
H3:    0 <= integer__last .
H4:    fld_size_of_stack(s) = 0 .
```

```
H5:    integer__size >= 0 .
H6:    integer__first <= integer__last .
H7:    integer__base__first <= integer__base__last .
H8:    integer__base__first <= integer__first .
H9:    integer__base__last >= integer__last .
H10:   stack__size >= 0 .
H11:   pointer_stack__size >= 0 .
H12:   pointer_stack__base__first <= pointer_stack__base__last .
H13:   index_stack__size >= 0 .
H14:   index_stack__base__first <= index_stack__base__last .
H15:   pointer_stack__base__first <= 0 .
H16:   pointer_stack__base__last >= 10 .
H17:   index_stack__base__first <= 1 .
H18:   index_stack__base__last >= 10 .
       ->
C1:    0 = element(fld_data_in_stack(s), [fld_size_of_stack(s)]) .


For checks of refinement integrity:

function_top_4.
*** true .          /* all conclusions proved */


function_top_5.
*** true .          /* all conclusions proved */


******************************************************************************
                     Semantic Analysis of SPARK Text
                            Examiner GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
******************************************************************************


CREATED 16-JUN-2016, 18:05:50  SIMPLIFIED 16-JUN-2016, 18:06:14

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

function Stack2005.Empty




For path(s) from start to finish:

function_empty_1.
*** true .          /* all conclusions proved */


For checks of refinement integrity:

function_empty_2.
*** true .          /* all conclusions proved */


function_empty_3.
*** true .          /* all conclusions proved */



******************************************************************************
```

```
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
************************************************************************


CREATED 16-JUN-2016, 18:05:50  SIMPLIFIED 16-JUN-2016, 18:06:14

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

function Stack2005.Full




For path(s) from start to finish:

function_full_1.
*** true .          /* all conclusions proved */


For checks of refinement integrity:

function_full_2.
*** true .          /* all conclusions proved */


function_full_3.
*** true .          /* all conclusions proved */
```

Conteúdo dos ficheiros Push.siv, Pop.siv, Clear.siv, Top.siv, Empty.siv e Full.siv

Anexo V

```
            ******************************************************
                        Semantic Analysis of SPARK Text
                               Examiner GPL 2012
              Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
            ******************************************************


                          DATE : 19-MAR-2016 10:16:38.11

                       function Factorial2005.Factorial




For path(s) from start to run-time check associated with statement of line 7:

function_factorial_1.
H1:    n >= 0 .
H2:    n <= 12 .
H3:    n >= natural__first .
H4:    n <= natural__last .
        ->
C1:    n >= integer__first .
C2:    n <= integer__last .
C3:    1 >= integer__first .
C4:    1 <= integer__last .


For path(s) from start to run-time check associated with statement of line 7:

function_factorial_2.
H1:    n >= 0 .
H2:    n <= 12 .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= integer__first .
H6:    n <= integer__last .
H7:    1 >= integer__first .
H8:    1 <= integer__last .
        ->
C1:    (1 <= n) -> ((n >= natural__first) and (n <= natural__last)) .
C2:    (1 <= n) -> ((1 >= natural__first) and (1 <= natural__last)) .


For path(s) from start to run-time check associated with statement of line 9:

function_factorial_3.
H1:    n >= 0 .
H2:    n <= 12 .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= integer__first .
H6:    n <= integer__last .
H7:    1 >= integer__first .
H8:    1 <= integer__last .
H9:    (1 <= n) -> ((n >= natural__first) and (n <= natural__last)) .
H10:   (1 <= n) -> ((1 >= natural__first) and (1 <= natural__last)) .
H11:   1 <= n .
H12:   not (1 > n) .
H13:   1 >= natural__first .
```

```
H14:   1 <= natural__last .
        ->
C1:    1 * 1 >= natural__first .
C2:    1 * 1 <= natural__last .


For path(s) from assertion of line 10 to run-time check associated with statement of
line 9:

function_factorial_4.
H1:    loop__1__i > 0 .
H2:    result = fact(loop__1__i) .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= 0 .
H6:    n <= 12 .
H7:    loop__1__i >= natural__first .
H8:    loop__1__i <= natural__last .
H9:    loop__1__i >= 1 .
H10:   loop__1__i <= n__entry__loop__1 .
H11:   n__entry__loop__1 >= natural__first .
H12:   n__entry__loop__1 <= natural__last .
H13:   not (loop__1__i = n__entry__loop__1) .
H14:   not (loop__1__i + 1 > n) .
H15:   result >= natural__first .
H16:   result <= natural__last .
        ->
C1:    result * (loop__1__i + 1) >= natural__first .
C2:    result * (loop__1__i + 1) <= natural__last .


For path(s) from start to assertion of line 10:

function_factorial_5.
H1:    n >= 0 .
H2:    n <= 12 .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= integer__first .
H6:    n <= integer__last .
H7:    1 >= integer__first .
H8:    1 <= integer__last .
H9:    (1 <= n) -> ((n >= natural__first) and (n <= natural__last)) .
H10:   (1 <= n) -> ((1 >= natural__first) and (1 <= natural__last)) .
H11:   1 <= n .
H12:   not (1 > n) .
H13:   1 >= natural__first .
H14:   1 <= natural__last .
H15:   1 * 1 >= natural__first .
H16:   1 * 1 <= natural__last .
        ->
C1:    1 > 0 .
C2:    1 * 1 = fact(1) .
C3:    n >= natural__first .
C4:    n <= natural__last .
C5:    n >= 0 .
C6:    n <= 12 .
C7:    1 >= natural__first .
C8:    1 <= natural__last .
C9:    1 >= 1 .
C10:   1 <= n .
C11:   n >= natural__first .
C12:   n <= natural__last .
```

```
For path(s) from assertion of line 10 to assertion of line 10:

function_factorial_6.
H1:    loop__1__i > 0 .
H2:    result = fact(loop__1__i) .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= 0 .
H6:    n <= 12 .
H7:    loop__1__i >= natural__first .
H8:    loop__1__i <= natural__last .
H9:    loop__1__i >= 1 .
H10:   loop__1__i <= n__entry__loop__1 .
H11:   n__entry__loop__1 >= natural__first .
H12:   n__entry__loop__1 <= natural__last .
H13:   not (loop__1__i = n__entry__loop__1) .
H14:   not (loop__1__i + 1 > n) .
H15:   result >= natural__first .
H16:   result <= natural__last .
H17:   result * (loop__1__i + 1) >= natural__first .
H18:   result * (loop__1__i + 1) <= natural__last .
        ->
C1:    loop__1__i + 1 > 0 .
C2:    result * (loop__1__i + 1) = fact(loop__1__i + 1) .
C3:    n >= natural__first .
C4:    n <= natural__last .
C5:    n >= 0 .
C6:    n <= 12 .
C7:    loop__1__i + 1 >= natural__first .
C8:    loop__1__i + 1 <= natural__last .
C9:    loop__1__i + 1 >= 1 .
C10:   loop__1__i + 1 <= n__entry__loop__1 .
C11:   n__entry__loop__1 >= natural__first .
C12:   n__entry__loop__1 <= natural__last .


For path(s) from start to finish:

function_factorial_7.
H1:    n >= 0 .
H2:    n <= 12 .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= integer__first .
H6:    n <= integer__last .
H7:    1 >= integer__first .
H8:    1 <= integer__last .
H9:    (1 <= n) -> ((n >= natural__first) and (n <= natural__last)) .
H10:   (1 <= n) -> ((1 >= natural__first) and (1 <= natural__last)) .
H11:   not (1 <= n) .
H12:   1 >= natural__first .
H13:   1 <= natural__last .
        ->
C1:    1 = fact(n) .
C2:    1 >= natural__first .
C3:    1 <= natural__last .


function_factorial_8.
H1:    n >= 0 .
H2:    n <= 12 .
```

```
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= integer__first .
H6:    n <= integer__last .
H7:    1 >= integer__first .
H8:    1 <= integer__last .
H9:    (1 <= n) -> ((n >= natural__first) and (n <= natural__last)) .
H10:   (1 <= n) -> ((1 >= natural__first) and (1 <= natural__last)) .
H11:   1 <= n .
H12:   1 > n .
H13:   1 >= natural__first .
H14:   1 <= natural__last .
        ->
C1:    1 = fact(n) .
C2:    1 >= natural__first .
C3:    1 <= natural__last .

For path(s) from assertion of line 10 to finish:
function_factorial_9.
H1:    loop__1__i > 0 .
H2:    result = fact(loop__1__i) .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= 0 .
H6:    n <= 12 .
H7:    loop__1__i >= natural__first .
H8:    loop__1__i <= natural__last .
H9:    loop__1__i >= 1 .
H10:   loop__1__i <= n__entry__loop__1 .
H11:   n__entry__loop__1 >= natural__first .
H12:   n__entry__loop__1 <= natural__last .
H13:   loop__1__i = n__entry__loop__1 .
H14:   result >= natural__first .
H15:   result <= natural__last .
        ->
C1:    result = fact(n) .
C2:    result >= natural__first .
C3:    result <= natural__last .

function_factorial_10.
H1:    loop__1__i > 0 .
H2:    result = fact(loop__1__i) .
H3:    n >= natural__first .
H4:    n <= natural__last .
H5:    n >= 0 .
H6:    n <= 12 .
H7:    loop__1__i >= natural__first .
H8:    loop__1__i <= natural__last .
H9:    loop__1__i >= 1 .
H10:   loop__1__i <= n__entry__loop__1 .
H11:   n__entry__loop__1 >= natural__first .
H12:   n__entry__loop__1 <= natural__last .
H13:   not (loop__1__i = n__entry__loop__1) .
H14:   loop__1__i + 1 > n .
H15:   result >= natural__first .
H16:   result <= natural__last .
        ->
C1:    result = fact(n) .
C2:    result >= natural__first .
C3:    result <= natural__last .
```

Conteúdo do ficheiro Factorial2005.vcg

Anexo VI

```
******************************************************************************
                      Semantic Analysis of SPARK Text
                             Examiner GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
******************************************************************************


CREATED 19-MAR-2016, 10:16:38  SIMPLIFIED 19-MAR-2016, 10:16:43

SPARK Simplifier GPL 2012
Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.

function Factorial2005.Factorial




For path(s) from start to run-time check associated with statement of line 7:

function_factorial_1.
H1:    n >= 0 .
H2:    n <= 12 .
H3:    n <= natural__last .
H4:    integer__size >= 0 .
H5:    integer__first <= integer__last .
H6:    integer__base__first <= integer__base__last .
H7:    integer__base__first <= integer__first .
H8:    integer__base__last >= integer__last .
H9:    natural__size >= 0 .
H10:   natural__base__first <= natural__base__last .
H11:   natural__base__last >= natural__last .
H12:   0 <= natural__last .
H13:   natural__base__first <= 0 .
       ->
C1:    n >= integer__first .
C2:    n <= integer__last .
C3:    1 >= integer__first .
C4:    1 <= integer__last .


For path(s) from start to run-time check associated with statement of line 7:

function_factorial_2.
*** true .          /* all conclusions proved */


For path(s) from start to run-time check associated with statement of line 9:

function_factorial_3.
*** true .          /* all conclusions proved */


For path(s) from assertion of line 10 to run-time check associated with
          statement of line 9:

function_factorial_4.
H1:    loop__1__i > 0 .
H2:    n <= natural__last .
H3:    n <= 12 .
H4:    n__entry__loop__1 <= natural__last .
```

```
H5:    loop__1__i + 1 <= n .
H6:    fact(loop__1__i) >= 0 .
H7:    fact(loop__1__i) <= natural__last .
H8:    loop__1__i < n__entry__loop__1 .
H9:    integer__size >= 0 .
H10:   integer__first <= integer__last .
H11:   integer__base__first <= integer__base__last .
H12:   integer__base__first <= integer__first .
H13:   integer__base__last >= integer__last .
H14:   natural__size >= 0 .
H15:   natural__base__first <= natural__base__last .
H16:   natural__base__last >= natural__last .
H17:   0 <= natural__last .
H18:   natural__base__first <= 0 .
       ->
C1:    fact(loop__1__i) * (loop__1__i + 1) <= natural__last .


For path(s) from start to assertion of line 10:

function_factorial_5.
H1:    n <= 12 .
H2:    n <= natural__last .
H3:    n >= integer__first .
H4:    n <= integer__last .
H5:    1 >= integer__first .
H6:    1 <= integer__last .
H7:    1 <= natural__last .
H8:    1 <= n .
H9:    1 <= natural__last .
H10:   integer__size >= 0 .
H11:   integer__first <= integer__last .
H12:   integer__base__first <= integer__base__last .
H13:   integer__base__first <= integer__first .
H14:   integer__base__last >= integer__last .
H15:   natural__size >= 0 .
H16:   natural__base__first <= natural__base__last .
H17:   natural__base__last >= natural__last .
H18:   0 <= natural__last .
H19:   natural__base__first <= 0 .
       ->
C1:    1 = fact(1) .


For path(s) from assertion of line 10 to assertion of line 10:

function_factorial_6.
H1:    loop__1__i > 0 .
H2:    n <= natural__last .
H3:    n <= 12 .
H4:    n__entry__loop__1 <= natural__last .
H5:    loop__1__i + 1 <= n .
H6:    fact(loop__1__i) >= 0 .
H7:    fact(loop__1__i) <= natural__last .
H8:    fact(loop__1__i) * (loop__1__i + 1) >= 0 .
H9:    fact(loop__1__i) * (loop__1__i + 1) <= natural__last .
H10:   loop__1__i < n__entry__loop__1 .
H11:   integer__size >= 0 .
H12:   integer__first <= integer__last .
H13:   integer__base__first <= integer__base__last .
H14:   integer__base__first <= integer__first .
H15:   integer__base__last >= integer__last .
H16:   natural__size >= 0 .
```

```
H17:   natural__base__first <= natural__base__last .
H18:   natural__base__last >= natural__last .
H19:   0 <= natural__last .
H20:   natural__base__first <= 0 .
       ->
C1:    fact(loop__1__i) * (loop__1__i + 1) = fact(loop__1__i + 1) .


For path(s) from start to finish:

function_factorial_7.
H1:    n >= 0 .
H2:    n <= natural__last .
H3:    n >= integer__first .
H4:    n <= integer__last .
H5:    1 <= integer__last .
H6:    1 <= n -> n >= 0 and n <= natural__last .
H7:    1 <= n -> 1 <= natural__last .
H8:    n < 1 .
H9:    1 <= natural__last .
H10:   integer__size >= 0 .
H11:   integer__first <= integer__last .
H12:   integer__base__first <= integer__base__last .
H13:   integer__base__first <= integer__first .
H14:   integer__base__last >= integer__last .
H15:   natural__size >= 0 .
H16:   natural__base__first <= natural__base__last .
H17:   natural__base__last >= natural__last .
H18:   0 <= natural__last .
H19:   natural__base__first <= 0 .
       ->
C1:    1 = fact(n) .


function_factorial_8.
*** true .   /* contradiction within hypotheses. */



For path(s) from assertion of line 10 to finish:

function_factorial_9.
H1:    n__entry__loop__1 > 0 .
H2:    n >= 0 .
H3:    n <= natural__last .
H4:    n <= 12 .
H5:    n__entry__loop__1 <= natural__last .
H6:    n__entry__loop__1 >= 0 .
H7:    n__entry__loop__1 <= natural__last .
H8:    fact(n__entry__loop__1) >= 0 .
H9:    fact(n__entry__loop__1) <= natural__last .
H10:   integer__size >= 0 .
H11:   integer__first <= integer__last .
H12:   integer__base__first <= integer__base__last .
H13:   integer__base__first <= integer__first .
H14:   integer__base__last >= integer__last .
H15:   natural__size >= 0 .
H16:   natural__base__first <= natural__base__last .
H17:   natural__base__last >= natural__last .
H18:   0 <= natural__last .
H19:   natural__base__first <= 0 .
       ->
C1:    fact(n__entry__loop__1) = fact(n) .
```

```
function_factorial_10.
H1:    loop__1__i > 0 .
H2:    n >= 0 .
H3:    n <= 12 .
H4:    n__entry__loop__1 <= natural__last .
H5:    loop__1__i + 1 > n .
H6:    fact(loop__1__i) >= 0 .
H7:    fact(loop__1__i) <= natural__last .
H8:    loop__1__i < n__entry__loop__1 .
H9:    integer__size >= 0 .
H10:   integer__first <= integer__last .
H11:   integer__base__first <= integer__base__last .
H12:   integer__base__first <= integer__first .
H13:   integer__base__last >= integer__last .
H14:   natural__size >= 0 .
H15:   natural__base__first <= natural__base__last .
H16:   natural__base__last >= natural__last .
H17:   0 <= natural__last .
H18:   natural__base__first <= 0 .
       ->
C1:    fact(loop__1__i) = fact(n) .
```

Conteúdo do ficheiro Factorial2005.siv

Anexo VII

Pilha (Stack)

Configuração projecto

```
project Stack2005 is

   for Languages use ("Ada");
   for Source_Dirs use ("./**");
   for Main use ("stack2005_Teste.adb");
   for Source_Files use ("Stack2005.adb", "Stack2005.ads", "stack2005_Teste.adb");

   package Naming is
      for Spec_Suffix ("metafile") use ".smf";
      for Spec_Suffix ("siv") use ".siv";
      for Spec_Suffix ("vcg") use ".vcg";
      for Spec_Suffix ("pogs summary") use ".sum";
      for Spec_Suffix ("rlu") use ".rlu";
      for Spec_Suffix ("prv") use ".prv";
      for Spec_Suffix ("plg") use ".plg";
      for Spec_Suffix ("rul") use ".rul";
      for Spec_Suffix ("rls") use ".rls";
      for Spec_Suffix ("fdl") use ".fdl";
      for Spec_Suffix ("slg") use ".slg";
      for Spec_Suffix ("cmd") use ".cmd";
      for Spec_Suffix ("dpc") use ".dpc";
      for Spec_Suffix ("sdp") use ".sdp";
      for Spec_Suffix ("vct") use ".vct";
      for Spec_Suffix ("vlg") use ".vlg";
      for Spec_Suffix ("index") use ".idx";
      for Spec_Suffix ("listing") use ".lss";
      for Body_Suffix ("listing") use ".lsb";
   end Naming;

   package Ide is
      for Default_Switches ("examiner") use ("-warning_file=Stack2005.wrn", "-noswitch", "-
index_file=Stack2005", "-flow_analysis=information", "-vcg", "-dpc", "-listing=LST", "-
report=Stack2005.REP", "-language=2005",
         "-annotation_character=#");
      for Default_Switches ("sparksimp") use ("-a", "-e");
      for Default_Switches ("simplifier") use ("-plain");
      for Default_Switches ("victor") use ("-plain");
      for Default_Switches ("riposte") use ("--reference");
      for Default_Switches ("zombiescope") use ("-plain");
      for Default_Switches ("sparkmake") use ("-
dir=D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2005");
      for Default_Switches ("pogs") use ("-o=Stac2005.POGS");
   end Ide;

   package Compiler is
      for Default_Switches ("ada") use ("-gnato", "-ftest-coverage", "-g", "-gnata", "-
fprofile-arcs", "-gnat05", "-gnat95");
   end Compiler;

   package Linker is
      for Default_Switches ("ada") use ("-fprofile-generate", "-g");
   end Linker;

   package Prove is
      for Switches use ("--report=all");
   end Prove;

   package Builder is
      for Default_Switches ("ada") use ("-g");
   end Builder;

   package Pretty_Printer is
```

```
      for Default_Switches ("ada") use ("-kU");
   end Pretty_Printer;

end Stack2005;
```

Stack2005.ads

```
--SPARK 2005
package Stack2005
is
    type Stack is private;

    function Top (S : in Stack) return Integer;
    function Empty (S : in Stack) return Boolean;
    function Full (S : in Stack) return Boolean;

    procedure Push (S : in out Stack; I : in Integer);
    --# pre not Full(S);
    --# post not Empty(S);

    procedure Pop (S : in out Stack; I : out Integer);
    --# pre not Empty(S);
    --# post not Full(S);
    procedure Clear (S : in out Stack);
    --# post Empty(S);

private
    Stack_Size : constant := 10; -- tamanho maximo da stack
    type Pointer_Stack is range 0 .. Stack_Size;
    subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
    type Vector is array (Index_Stack) of Integer;

    type Stack is record
        Size_of_Stack : Pointer_Stack;  --:= 0;
        Data_in_Stack : Vector; --:= Vector'(Index_Stack => 0);
    end record;

end Stack2005;
```

Stack2005.adb.

```
--SPARK 2005
package body Stack2005
is
    procedure Push (S : in out Stack; I : Integer)
    is
    begin
        S.Size_of_stack := S.Size_of_stack + 1;
        S.Data_in_stack (S.Size_of_stack) := I;
    end Push;

    procedure Pop (S : in out Stack; I : out Integer)
    is
    begin
        I := S.Data_in_stack (S.Size_of_stack);
        S.Size_of_stack := S.Size_of_stack - 1;
    end Pop;

    procedure Clear (S : in out Stack)
    is
    begin
        S.Size_of_Stack := 0;
    end Clear;

    function Top (S : in Stack) return Integer
```

```
    --# return S.Data_in_stack (S.Size_of_stack);
    is
        v_top : integer := 0;
    begin
        if S.Size_of_stack > 0 then
            v_top := S.Data_in_stack (S.Size_of_stack);
        end if;
        return v_top;
    end Top;

    function Empty (S : in Stack) return Boolean
    --# return S.Size_of_Stack = 0;
    is
    begin
        return S.Size_of_Stack = 0;
    end Empty;

    function Full (S : in Stack) return Boolean
    --# return S.Size_of_Stack = Stack_Size;
    is
    begin
        return S.Size_of_Stack = Stack_Size;
    end Full;

end Stack2005;
```

Stack2005_Teste.adb

```
--
with Gnat.Io; use Gnat.Io;
with Stack2005; use Stack2005;

procedure Stack2005_Teste is

    S : Stack;
    Input: Integer;
    n:constant:= 0;

begin
    -- Carrega alguns inteiros na stack
    while not Full(S) loop
        Put("Digite um numero: ");
        Get(Input);
        exit when Full(S) or Input = -1;
        Push(S,Input);
    end loop;

    -- descarrega a stack e mostra no ecran
    while not Empty(S) loop
        Put("Retirado da STACK: ");
        Pop(S,Input);
        Put(Input);
        Put_Line(" ");
        exit when Empty(S);
    end loop;
    New_Line;
end Stack2005_Teste;
```

Output de Stack2005_Teste

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2005>D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2005
\stack2
005_teste.exe
Digite um numero: 1
Digite um numero: 2
```

```
Digite um numero: 3
Digite um numero: 4
Digite um numero: 5
Digite um numero: 67
Digite um numero: 8
Digite um numero: 9
Digite um numero: 10
Digite um numero: 11
Retirado da STACK: 11
Retirado da STACK: 10
Retirado da STACK: 9
Retirado da STACK: 8
Retirado da STACK: 67
Retirado da STACK: 5
Retirado da STACK: 4
Retirado da STACK: 3
Retirado da STACK: 2
Retirado da STACK: 1


D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack\Stack2005>
```

Lista Circular (CircularBuffer)

Configuração projecto

```
project ListaCircular2014 is

   for Languages use ("Ada", "Cmd", "Dpc", "Fdl", "Index", "Listing", "Metafile", "Plg",
"Pogs Summary", "Prv", "Rls", "Rlu", "Rul", "Sdp", "Siv", "Slg", "Vcg", "Vct", "Vlg");
   for Source_Dirs use ("./**");
   for Main use ("ListaCircular2014_io.adb");
   for Object_Dir use ".\obj";

   package Naming is
      for Spec_Suffix ("metafile") use ".smf";
      for Spec_Suffix ("siv") use ".siv";
      for Spec_Suffix ("vcg") use ".vcg";
      for Spec_Suffix ("pogs summary") use ".sum";
      for Spec_Suffix ("rlu") use ".rlu";
      for Spec_Suffix ("prv") use ".prv";
      for Spec_Suffix ("plg") use ".plg";
      for Spec_Suffix ("rul") use ".rul";
      for Spec_Suffix ("rls") use ".rls";
      for Spec_Suffix ("fdl") use ".fdl";
      for Spec_Suffix ("slg") use ".slg";
      for Spec_Suffix ("cmd") use ".cmd";
      for Spec_Suffix ("dpc") use ".dpc";
      for Spec_Suffix ("sdp") use ".sdp";
      for Spec_Suffix ("vct") use ".vct";
      for Spec_Suffix ("vlg") use ".vlg";
      for Spec_Suffix ("index") use ".idx";
      for Spec_Suffix ("listing") use ".lss";
      for Body_Suffix ("listing") use ".lsb";
   end Naming;

   package Ide is
      for Default_Switches ("examiner") use ("-noswitch", "-
index_file=ListaCircular2014.idx", "-warning_file=ListaCircular2014.wrn", "-language=2005",
"-flow_analysis=information", "-vcg", "-syntax_check");
      for Default_Switches ("sparksimp") use ("-a", "-e");
      for Default_Switches ("simplifier") use ("-plain");
      for Default_Switches ("victor") use ("-plain");
      for Default_Switches ("riposte") use ("--reference");
      for Default_Switches ("zombiescope") use ("-plain");
      for Default_Switches ("sparkmake") use ("-
dir=D:\DISSERTACAO\DEMO\2014\ListaCircular2014");
      for Default_Switches ("pogs") use ("-p");
   end Ide;

   package Compiler is
      for Default_Switches ("ada") use ("-gnatVa", "-gnat12", "-gnato", "-ftest-coverage", "-
g", "-gnata", "-fprofile-arcs", "-O2", "-gnatQ", "-fstack-check");
   end Compiler;

   package Linker is
      for Default_Switches ("ada") use ("-fprofile-generate", "-g");
   end Linker;

   package Prove is
      for Switches use ("--report=all", "--steps=1");
   end Prove;

   package Builder is
      for Default_Switches ("ada") use ("-g");
   end Builder;

   package Pretty_Printer is
      for Default_Switches ("ada") use ("-kU");
   end Pretty_Printer;

end ListaCircular2014;
```

ListaCircular2014.ads

```
package ListaCircular2014
with SPARK_Mode
is

    type ListaCircular is private;

    procedure Guardar (CB : in out ListaCircular; Inteiro : in Integer)
        with
            Pre => (not ListaCheia (CB));

    procedure Ler (CB : in out ListaCircular; Inteiro : out Integer)
        with
            pre => ( not ListaVazia (CB));

    function ListaCheia (CB : in ListaCircular) return Boolean;
    function ListaVazia (CB : in ListaCircular) return boolean;

private
    ListaCirc_Size : constant := 10;
    type Cauda_ListaCirc is range 0 .. ListaCirc_Size;
    subtype Index_Cauda_ListaCirc
        is Cauda_ListaCirc range 1 .. ListaCirc_Size;
    type Vector is array (Index_Cauda_ListaCirc) of Integer;

    type ListaCircular is record
        Cabeca : Cauda_ListaCirc := 0;
        Cauda  : Cauda_ListaCirc := 0;
        Dados  : Vector := Vector'(Index_Cauda_ListaCirc => 0);
    end record;
end ListaCircular2014;
```

ListaCircula2014.adb.

```
package body ListaCircular2014
with SPARK_Mode
is

    procedure Guardar (CB : in out ListaCircular; Inteiro : in Integer)
    is
    begin
        if CB.Cauda < ListaCirc_Size then
            pragma assert (not ListaCheia (CB));
            CB.Cauda := CB.Cauda + 1;
            pragma assert (if CB.Cauda < ListaCirc_Size then CB.Cauda + 1 <=
ListaCirc_Size );
            CB.Dados (CB.Cauda) := Inteiro;
            if CB.Cauda = CB.Dados'Length then
                CB.Cauda := 0;
            end if;
        end if;
    end Guardar;



    procedure Ler (CB : in out ListaCircular; Inteiro : out Integer) is
    begin
        Inteiro := 0;
--      if CB.Cabeca /= CB.Cauda then
        if CB.Cabeca < ListaCirc_Size then
            pragma assert (not ListaVazia (CB));
            CB.Cabeca := CB.Cabeca + 1;
            pragma assert(if CB.Cabeca < ListaCirc_Size then CB.Cabeca + 1 <=
ListaCirc_Size);
            Inteiro := CB.Dados (CB.Cabeca);
```

```
            if CB.Cabeca = CB.Dados'Length then
                CB.Cabeca := 0;
            end if;
        end if;
    end Ler;



    function ListaCheia (CB : in ListaCircular) return Boolean is
        cheia : Boolean := False;
    begin
        if CB.Cauda + 1 = CB.Cabeca then
            cheia := true;
        elsif CB.Cauda = CB.Dados'Length - 1 and CB.Cabeca = 0 then
            cheia := true;
        end if;
        return cheia;
    end ListaCheia;


    function ListaVazia (CB : in ListaCircular) return boolean is
        vazia : Boolean := False;
    begin
        if CB.Cabeca = 0 and CB.Cauda = 0 then
            vazia := true;
        end if;
        return vazia;
    end ListaVazia;

end ListaCircular2014;
```

ListaCircular_io.adb

```
--
with Gnat.Io; use Gnat.Io;
with ListaCircular2014; use ListaCircular2014;

procedure ListaCircular2014_io is

        circbuff : ListaCircular;
        inteiro : Integer := 0;
begin



        Put_Line ("Guardar: 1");
        Guardar(circbuff, 1);
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put_Line("Guardar: 2");
        Guardar(circbuff, 2);
        Put_Line("Guardar: 3");
        Guardar(circbuff, 3);
        Put_Line("Guardar: 4");
        Guardar(circbuff, 4);
        Put("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put_Line("Guardar: 8");
        Guardar(circbuff, 8);
        Put_Line("Guardar: 9");
        Guardar(circbuff, 9);
```

```
        Put_Line("Guardar: 10");
        Guardar(circbuff, 10);
        Put_Line("Guardar: 11");
        Guardar(circbuff, 11);
        Put_Line("Guardar: 12");
        Guardar(circbuff, 12);
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");
        Put ("Ler: ");
        Ler (circbuff, inteiro);
        put (inteiro);
        Put_Line (" ");

end ListaCircular2014_io
```

Output de ListaCircular_io

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\ListaCircular\ListaCircular2014>D:\_DISSERTACAO\_EXEMPLOS_TESE
\ListaC
ircular\ListaCircular2014\obj\listacircular2014_io.exe
Guardar: 1
Ler: 1
Guardar: 2
Guardar: 3
Guardar: 4
Ler: 2
Ler: 3
Guardar: 8
Guardar: 9
Guardar: 10
Guardar: 11
Guardar: 12
Ler: 4
Ler: 8
Ler: 9
Ler: 10
Ler: 11
Ler: 12
Ler: 0
Ler: 1D:\_DISSERTACAO\_EXEMPLOS_TESE\ListaCircular\ListaCircular2014>
```

Fatorial

Configuração projecto Fatorial2005

```
project Factorial2005 is

   for Languages use ("Ada");
   for Source_Dirs use ("./**");
   for Source_Files use ("factorial_teste.adb", "factorial2005.adb", "factorial2005.ads");
   for Object_Dir use "obj";
   for Exec_Dir use "exec";
   for Main use ("factorial_teste.adb");

   package Naming is
      for Spec_Suffix ("metafile") use ".smf";
      for Spec_Suffix ("siv") use ".siv";
      for Spec_Suffix ("vcg") use ".vcg";
      for Spec_Suffix ("pogs summary") use ".sum";
      for Spec_Suffix ("rlu") use ".rlu";
      for Spec_Suffix ("prv") use ".prv";
      for Spec_Suffix ("plg") use ".plg";
      for Spec_Suffix ("rul") use ".rul";
      for Spec_Suffix ("rls") use ".rls";
      for Spec_Suffix ("fdl") use ".fdl";
      for Spec_Suffix ("slg") use ".slg";
      for Spec_Suffix ("cmd") use ".cmd";
      for Spec_Suffix ("dpc") use ".dpc";
      for Spec_Suffix ("sdp") use ".sdp";
      for Spec_Suffix ("vct") use ".vct";
      for Spec_Suffix ("index") use ".idx";
      for Spec_Suffix ("listing") use ".lss";
      for Body_Suffix ("listing") use ".lsb";
   end Naming;

   package Compiler is
      for Default_Switches ("ada") use ("-gnato", "-fstack-check", "-gnat05");
   end Compiler;

   package Ide is
      for Default_Switches ("sparksimp") use ("-a", "-l", "-v", "-e");
      for Default_Switches ("simplifier") use ("-plain", "-noecho");
      for Default_Switches ("victor") use ("-plain");
      for Default_Switches ("pogs") use ("-p");
      for Default_Switches ("examiner") use ("-language=2005", "-flow_analysis=information",
"-vcg", "-dpc",
         "-index_file=D:\_DISSERTACAO\DEMO\Factorial\Factorial2005\Factorial2005.idx", "-
error_explanations=every", "-listing=lse", "-report=Factorial2005");
      for Default_Switches ("zombiescope") use ("-plain");
      for Default_Switches ("sparkformat") use ("-global_indent=4", "-separator_indent=4", "-
refinement_indent=4", "-properties_indent=4", "-export_indent=4", "-inherit_indent=4", "-
constituent_indent=4", "-import_indent=4", "-own_indent=4",
         "-initialization_indent=4");
   end Ide;

   package Pretty_Printer is
      for Default_Switches ("ada") use ("-i4", "-cl4");
   end Pretty_Printer;

end Factorial2005;
```

Fatorial2005.ads

```
package Factorial2005 is
        --# function Fact(N : Natural) return Natural;
        function Factorial (N : Natural) return Natural;
        --# pre (N >= 0 and N <= 12);
        --# return Fact(N);

end Factorial2005;
```

Fatorial2005.adb

```
package body Factorial2005 is
    --# function Fact(N : Natural) return Natural;
    function Factorial (N : Natural) return Natural
    is
        Result : Natural := 1;
    begin
        for I in Natural range 1 .. N loop
                exit when I > N;
                Result := Result * I;
                --#assert I > 0 and Result = Fact(I);
        end loop;
    return Result;
end Factorial;

end Factorial2005;
```

Fatorial_teste.adb

```
with Gnat.Io; --use Gnat.Io;
with Factorial2005; use Factorial2005;

procedure Factorial_Teste is

    Result,Input: Integer;

begin
        Gnat.Io.Put("Digite um numero: ");
        Gnat.Io.Get(Input);
        Result := Factorial2005.Factorial(Input);

        Gnat.Io.Put (Input);
        Gnat.Io.Put ("! = ");
        Gnat.Io.Put(Result);
        Gnat.Io.Put_Line(" ");
        Gnat.Io.New_Line;
end Factorial_Teste;
```

Output de Fatorial_teste

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\Factorial\Factorial2005>D:\_DISSERTACAO\_EXEMPLOS_TESE\Factori
al\Fact
orial2005\exec\factorial_teste.exe
Digite um numero: 6
6! = 720


D:\_DISSERTACAO\_EXEMPLOS_TESE\Factorial\Factorial2005>
```

Configuração projecto Fatorial2014

```
project Factorial2014 is

   for Source_Dirs use ("./**");
   for Object_Dir use "obj";
   for Exec_Dir use "exec";
   for Main use ("factorial_teste.adb");

   package Compiler is
      for Default_Switches ("ada") use ("-gnat12", "-gnato", "-fstack-check", "-gnatwlu.y");
   end Compiler;

   for Source_Files use ("factorial_teste.adb", "factorial2014.adb", "factorial2014.ads");

   package Binder is
      for Default_Switches ("ada") use ("-r");
   end Binder;

end Factorial2014;
```

Fatorial2014.ads

```
package Factorial2014
with SPARK_Mode
is

    function Factorial (N : in Natural) return Natural
        with
        Depends => (Factorial'Result => N),
        pre => (N >= 0 and N <= 12),
        post => (Factorial'Result >= 0 and Factorial'Result - Natural'Last <= 0);

end Factorial2014;
```

Fatorial2014.adb

```
package body Factorial2014
with SPARK_Mode
is
    function Factorial (N : in Natural) return Natural
    is
        r     : Natural := 1;
        r_old : Natural := r with Ghost;
        i     : Natural := 0;

    begin
        loop
            pragma Loop_Invariant (I >= 0 and  I <= N);
            exit when i = N;
            i := i + 1;
            r_old := r;
            r := r * i;

            pragma Assert (if i <= N and r_old / N <= Natural'Last and r_old * i <=
Natural'Last then r <= Natural'Last);
            pragma Annotate (GNATprove, Intentional, "", "r always < Natural'Last because N<=
12");
        end loop;
        return r;
    end Factorial;

end Factorial2014;
```

Para testar este código pode-se utilizar o Fatorial_teste.adb. O output de execução é igual.

Versão com recurso a recursividade Fatorial2014R.ads

```
package Factorial2014R
with SPARK_Mode
is

    function FactorialR (N : in Natural) return Natural
        with
        Depends => (FactorialR'Result => N),
        pre => (N >= 0 and N <= 12),
        post => (FactorialR'Result >= 0 and FactorialR'Result - Natural'Last <= 0 and
FactorialR'Result <= Natural'Last);

end Factorial2014R;
```

Fatorial2014.adb

```
package body Factorial2014R
with SPARK_Mode
is
   function FactorialR (N : Natural) return Natural
   is
      r     : Natural := 0;
      r_old : Natural with Ghost;
   begin
      if n <= 1 then
         r := 1;
         pragma Assert (N < 2);
      elsif N <= 12 then
         pragma Assert ( N > 1 and N <= 12);
         r_old := r;
         pragma Assert (if N  >= 1 and N < 12 then r * (N-1) <= Natural'Last);
         r := N * FactorialR (N - 1);
         pragma Assert (if N  >= 1 and N < 12 and r / N <= Natural'Last and r / n =
FactorialR(N-1) and  FactorialR(N-1)<= Natural'Last then r <= Natural'Last);
        end if;
      return r;
   end FactorialR;
end Factorial2014R;
```

Para testar este código pode-se utilizar o Fatorial_teste.adb. O output de execução é igual.

NQUEENS

Esta versão só apresenta uma solução, o valor de número de rainhas está fixo no código.

Configuração projecto

```
project N_QUEENS is

   for Source_Dirs use ("./**");
   for Main use ("NQueens2014_IO.adb");
   for Source_Files use ("NQueens2014.adb", "NQueens2014.ads", "NQueens2014_IO.adb");
   for Object_Dir use "obj";
   for Exec_Dir use "exec";

   package Ide is
      for Artifacts_Dir use "artf";
   end Ide;

end N_QUEENS;
```

NQUEENS2014.ads

```
package NQueens2014
with SPARK_Mode

is
    type Tabuleiro is  array (Positive range <>, Positive range <>) of Boolean;
    Tab : Tabuleiro (1 .. 9, 1 .. 9) := (others => (others => False));

    function Testa (Linha, Coluna : Integer) return Boolean
        with
            pre => ((Linha >= 1 and Linha <= 9) and (Coluna >= 1 and Coluna <= 9))
    ;

    procedure Preenche (Coluna : Integer; Result : out Boolean)
        with
            pre => (Coluna >= 1 and Coluna <= 9)
    ;

end NQueens2014;
```

NQUEENS2014.adb.

```
package body NQueens2014
with SPARK_Mode

is
    function Testa (Linha, Coluna : Integer) return Boolean is
    begin
        for J in 1 .. Coluna - 1 loop
            pragma Loop_Invariant (Coluna >= 1 and Coluna <= 9);
            if (Tab (Linha, J)
            or else
                (Linha > J and then Tab (Linha - J, Coluna - J))
            or else
                (Linha + J <= 9 and then Tab (Linha + J, Coluna - J)))
            then
                return False;
            end if;
        end loop;
        return True;
```

```
    end Testa;

    procedure Preenche (Coluna : Integer; Result : out Boolean) --return Boolean
    is
    begin
        Result := False;

        for Linha in Tab'Range (1) loop
            pragma Loop_Invariant (Linha >= Tab'First (1) and Linha <= Tab'Last (1));
            if Testa (Linha, Coluna) then
                Tab (Linha, Coluna) := True;
                if Coluna = 9 then
                    Result := True;
                    return;
                else
                    Preenche (Coluna + 1, Result);
                    if Result = True then
                        return;
                    end if;
                end if;
                Tab (Linha, Coluna) := False;
                Result := False;
            end if;
        end loop;

    end Preenche;

end Nqueens2014;
```

NQueens2014_IO.adb

```
with Gnat.Io; use Gnat.Io;
with NQueens2014; use NQueens2014;

procedure NQueens2014_IO
is
    result : Boolean := False;
begin
    NQueens2014.Preenche (1, result);

   Put_Line ("  +-+-+-+-+-+-+-+-+-+");
   for I in Tab'Range (1) loop
      Put (Integer'Image (10 - I));
      for J in Tab'Range (2) loop
         if Tab (I, J) then
            Put ("|Q");
         elsif (I + J) mod 2 = 1 then
            Put ("|/");
         else
            Put ("| ");
         end if;
      end loop;
      Put_Line ("|");
   Put_Line ("  +-+-+-+-+-+-+-+-+-+");
   end loop;
   Put_Line ("   A B C D E F G H i ");--j k l m n o p q r s t u v w y");

end NQueens2014_IO;
```

Output de ListaCircular_io

```
D:\_DISSERTACAO\DEMO\N_QUEENS>D:\_DISSERTACAO\DEMO\N_QUEENS\exec\nqueens2014_io.exe
  +-+-+-+-+-+-+-+-+-+
 9|Q|/| |/| |/| |/| |
```

```
  +-+-+-+-+-+-+-+-+
8|/| |/| |Q| |/| |/|
  +-+-+-+-+-+-+-+-+
7| |Q| |/| |/| |/| |
  +-+-+-+-+-+-+-+-+
6|/| |/| |/|Q|/| |/|
  +-+-+-+-+-+-+-+-+
5| |/| |/| |/| |/|Q|
  +-+-+-+-+-+-+-+-+
4|/| |Q| |/| |/| |/|
  +-+-+-+-+-+-+-+-+
3| |/| |/| |/| |Q| |
  +-+-+-+-+-+-+-+-+
2|/| |/|Q|/| |/| |/|
  +-+-+-+-+-+-+-+-+
1| |/| |/| |/|Q|/| |
  +-+-+-+-+-+-+-+-+
   A B C D E F G H i

D:\_DISSERTACAO\DEMO\N_QUEENS>
```

Versão de NQUEENS que apresenta todas as soluções fundamentais para resolução do problema com o maximo de 20 rainhas.

Configuração projecto

```
project N_Queens2014 is

   for Source_Dirs use ("src/**");
   for Object_Dir use "obj";
   for Main use ("n_queens2014_io.adb");
   for Exec_Dir use "exec";
   for Source_Files use ("n_queens2014.adb", "n_queens2014.ads", "n_queens2014_io.adb",
"PrintSolution.adb");

   package Ide is
   end Ide;

   package Prove is
      for Switches use ("--level=0");
   end Prove;

end N_Queens2014;
```

NQUEENS2014.ads

```
with GNAT.IO; use GNAT.IO;

package N_Queens2014
with SPARK_Mode
is
    Max_Nrqueens     : constant := 20;
    MAx_Sol_Nrqueens : constant := 39029188884;
    ctd              : Long_Long_Integer := 0;
    NrQueens         : Natural := Max_Nrqueens;

    contador : Integer := 0;

    type Tabuleiro_tipo is array (1 .. NrQueens, 1 .. NrQueens)of Natural range 0 .. 1;
    Tabuleiro : Tabuleiro_tipo;
    type Apontador_tipo is array (1 .. NrQueens) of Natural range 0 .. Max_Nrqueens + 1;
    Apontador : Apontador_tipo;

    function TestaSolucao (Tabuleiro : in Tabuleiro_tipo) return Boolean
```

```
        with
            pre => (Tabuleiro'First (1) = 1 and Tabuleiro'Last (1) = NrQueens
             and Tabuleiro'First (2) = 1 and Tabuleiro'Last (2) = NrQueens)
            and (Tabuleiro'First (1) = 1 and Tabuleiro'Last (1) = NrQueens - 1
             and Tabuleiro'First (2) = 1 and Tabuleiro'Last (2) = NrQueens)
        ;

    procedure NQueens (N : in Natural)
        with
            pre => (N > 0 and N < Max_Nrqueens)
        and (Apontador'First = 1 and Apontador'Last = NrQueens),
        post => (for all i in Tabuleiro'Range (1) =>
                   (for all j in Tabuleiro'Range (2) => Tabuleiro (i, j) < 2))
            and (for all j in Apontador'Range => Apontador (j) >= 0
             and Apontador (j) <= (Max_Nrqueens + 1))
    ;

end N_Queens2014;
```

NQUEENS2014.adb.

```
with GNAT.IO; use GNAT.IO;
with PrintSolution;

package body N_Queens2014
with SPARK_Mode
is
    function TestaSolucao (Tabuleiro : in Tabuleiro_tipo) return Boolean
    is
        soma, p, q : Integer;
        idx_J      : Integer;

    begin

        for i in 1 .. Nrqueens loop
            soma := 1;
            for j in 1 .. Nrqueens loop
                soma := soma + Tabuleiro (i, j);
                if soma > 2 then
                    return False;
                end if;
            end loop;
        end loop;


        -- verifica diagonal
        idx_J := NrQueens - 1;
        while idx_J >= 1 loop
            soma := 1;
            p := 1;
            q := idx_J;
            while q <= NrQueens loop
                soma := soma + Tabuleiro (p, q);
                p := p + 1;
                q := q + 1;
            end loop;
            if soma > 2 then
                return False;
            end if;
            idx_J := idx_J - 1;
        end loop;
        for i in Integer range 2 .. NrQueens - 1 loop
            soma := 1;
            p := i;
            q := 1;
            while p <= NrQueens loop
                soma := soma + Tabuleiro (p, q);
                p := p + 1;
```

```
            q := q + 1;
        end loop;
        if soma > 2 then
            return False;
        end if;
    end loop;

    -- verifica diagonal
    for j in Integer range 2 .. NrQueens loop
        soma := 1;
        p := 1;
        q := j;
        while q >= 1 loop
            soma := soma + Tabuleiro (p, q);
            p := p + 1;
            q := q - 1;
        end loop;
        if soma > 2 then
            return False;
        end if;
    end loop;

    for i in Integer range 2 .. NrQueens - 1  loop
        soma := 1;
        p := i;
        q := NrQueens;
        while p <= NrQueens loop
            soma := soma + Tabuleiro (p, q);
            p := p + 1;
            q := q - 1;
        end loop;
        if soma > 2 then
            return False;
        end if;
    end loop;
    return true;
end TestaSolucao;

procedure NQueens (N : in Natural)
is
    ind : Integer := 1;
begin
    ctd := 0;
    NrQueens := N;
    pragma Assert (N <= Max_Nrqueens);

    while ind <= NrQueens loop
        pragma Loop_Invariant (ind > 0 and ind <= Nrqueens);
        Apontador (ind) := Apontador (ind) + 1;
        pragma Annotate (GNATprove, Intentional,
                  "", "ind e sempre <= Nrquens");
        if Apontador (ind) = NrQueens + 1 then
            Tabuleiro (Apontador (ind)-1, ind) := 0;
            pragma Annotate (GNATprove, Intentional,
               "", "Apontador (ind)-1 e sempre >= 1");
            Apontador (ind) := 0;
            ind := ind - 1;
            if ind = 0 then
                return;
            end if;
        else
            Tabuleiro (Apontador (ind), ind) := 1;
            pragma Annotate (GNATprove, Intentional,
               "", "Apontador (ind)-1 e sempre >= 1");
            if Apontador (ind) /= 1 then
                Tabuleiro (Apontador (ind) - 1, ind) := 0;
            end if;
            if TestaSolucao (Tabuleiro) then
                pragma Annotate (GNATprove, Intentional,
                   "", "Apontador (ind)-1 e sempre >= 1");
                ind := ind + 1;
```

```
                    if ind = NrQueens + 1 then
                        ind := ind - 1;
                        pragma Assume (ctd < Max_Sol_Nrqueens, "ctd e sempre <
Long_Long_Long_Integer");
                        ctd := ctd + 1;
                        PrintSolution (ctd, Nrqueens, Tabuleiro);
                    end if;
                end if;
            end if;
            exit when ind > NrQueens;
        end loop;

    end NQueens;

end N_Queens2014;
```

NQueens2014_IO.adb

```
with Gnat.Io; use Gnat.Io;
with N_Queens2014; use N_Queens2014;

procedure N_Queens2014_io
is
    Input : Natural := 1;

begin
    while Input > 0 loop
        Gnat.Io.Put ("Digite numero de Rainhas, 0 p/ fim: ");
        Gnat.Io.Get (Input);
        if Input < 1 then
            return;
        end if;
        if Input <= 20 then
            NQueens(Input);
        end if;
    end loop;
    --null;
end N_Queens2014_io;
```

PrintSolution.adb

```
with Gnat.Io; use Gnat.Io;
with N_Queens2014; use N_Queens2014;

procedure PrintSolution (solucao : in Long_Long_Integer; TamTab : in Integer; Tab : in
Tabuleiro_tipo)
is
begin
    Put ("Solucao: ");
    Put_Line (Long_Long_Integer'Image (solucao));
    for P in  1 .. TamTab loop
        for Q in  1 .. TamTab loop
            Put (Tab (P, Q));
            Put (" ");
        end loop;
        Put_Line ("");
    end loop;
end PrintSolution;
```

Output de NQueens2014_IO

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\N_Queens>D:\_DISSERTACAO\_EXEMPLOS_TESE\N_Queens\exec\n_queens
2014_io
.exe
Digite numero de Rainhas, 0 p/ fim: 5
Solucao:  1
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
Solucao:  2
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0
Solucao:  3
0 0 1 0 0
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
Solucao:  4
0 0 0 1 0
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0
Solucao:  5
0 1 0 0 0
0 0 0 1 0
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1
Solucao:  6
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
Solucao:  7
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0
0 0 0 1 0
Solucao:  8
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0
1 0 0 0 0
0 0 1 0 0
Solucao:  9
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
1 0 0 0 0
Solucao:  10
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0
1 0 0 0 0
Digite numero de Rainhas, 0 p/ fim:
```

Produtor/Consumidor

Configuração projecto

```
project Prod_Cons is

   for Source_Dirs use ("./**");
   for Main use ("Prod_Cons_Main.adb");
   --   for Object_Dir use "obj";
   --   for Exec_Dir use "exec";

   for Object_Dir use "./obj";
   for Exec_Dir use "./exec";
   for Source_Files use ("prod_cons.adb", "prod_cons.ads", "Prod_Cons_Main.adb");

   package Prove is
      for Switches use ("--steps=1", "--level=2");
   end Prove;

   package Compiler is
      for Default_Switches ("ada") use ("-gnat12", "-gnato", "-fstack-check", "-gnatE");
   end Compiler;

end Prod_Cons;
```

Prod_Cons.ads

```
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);

package Prod_Cons
with SPARK_Mode
is
   numero : Integer := 0;

   protected type Buffer is
      procedure PutValue (Val : in Integer)
        ;
      function GetValue return Integer
        ;
   private
      Num : Integer := 0;
   end Buffer;

   Nr     : Buffer;

   task Producer
     with
     Global   => (In_Out => Nr),
     Depends  => (Nr => Nr)
    ;

   task Consumer
     with
     Global   => (In_Out => Nr),
     Depends  => (Nr => Nr)
    ;

end Prod_Cons;
```

Prod_Cons.adb.

```
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);
with Gnat.IO; use GNAT.IO;

package body Prod_Cons
With SPARK_Mode
is

   protected body Buffer is
      procedure PutValue (Val : in Integer) is
      begin
         Num := Val;
      end PutValue;

      function GetValue return Integer is (Num);

   end Buffer;


   task body Producer is
   begin
      loop
         if numero < Integer'Last then
            numero := numero + 1;
         else
            numero := 0;
         end if;
         Nr.PutValue (numero);
         Put_Line ("PUT: " & Integer'Image (numero));
         pragma Annotate (GNATprove, Intentional,
                  "", "Put_Line foi adicionado somente p/mostrar valores. A retirar");
      end loop;
   end Producer;


   task body Consumer is
      My_Value     : Integer;
      My_Value_Old : Integer := 0;
   begin
      loop
         My_Value := Nr.GetValue;
         if My_Value_Old /= My_Value then
            My_Value_Old := My_Value;
            Put_Line ("GET: " & Integer'Image (My_Value) & " <<-----------");
            pragma Annotate (GNATprove, Intentional,
                     "", "Put_Line foi adicionado somente p/mostrar valores. A retirar");
         end if;
      end loop;
   end Consumer;

end Prod_Cons;
```

Prod_Cons_Main.adb

```
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);

with Prod_Cons; use Prod_Cons;
procedure Prod_Cons_Main
is
begin
   null;
end Prod_Cons_Main;
```

PrintProdCons.adb

```
with GNAT.IO; use GNAT.IO;

procedure PrintProdCons (str : in String)
is
begin
    Put_Line (str);
end PrintProdCons;
```

Output de Prod_Cons_Main

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese>D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod
_Cons_T
ese\exec\stackprodcons_io.exe
PushStack - Valor:  5000001 #########
PushStack - Valor:  5000002 #########
PushStack - Valor:  5000003 #########
PushStack - Valor:  5000004 #########
PushStack - Valor:  5000005 #########
PopStack - Valor:  5000005 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  5000004 ----------------------------
PopStack - SizeStack:  3
PopStack - Valor:  5000003 ----------------------------
PopStack - SizeStack:  2
PopStack - Valor:  5000002 ----------------------------
PopStack - SizeStack:  1
PopStack - Valor:  5000001 ----------------------------
PopStack - SizeStack:  0
PushStack - Valor:  5000006 #########
PushStack - Valor:  5000007 #########
PushStack - Valor:  5000008 #########
PushStack - Valor:  5000009 #########
PushStack - Valor:  5000010 #########
PopStack - Valor:  5000010 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  5000009 ----------------------------
PopStack - SizeStack:  3
PopStack - Valor:  5000008 ----------------------------
PopStack - SizeStack:  2
PopStack - Valor:  5000007 ----------------------------
PopStack - SizeStack:  1
PopStack - Valor:  5000006 ----------------------------
PopStack - SizeStack:  0
PushStack - Valor:  5000011 #########
PushStack - Valor:  5000012 #########
PushStack - Valor:  5000013 #########
PushStack - Valor:  5000014 #########
PushStack - Valor:  5000015 #########
PopStack - Valor:  5000015 ----------------------------
```

Stack Concorrente

Configuração projecto

```
project Stack_Prod_Cons is

   for Source_Dirs use ("src");
   for Object_Dir use "obj";
   for Main use ("stackprodcons_io.adb");
   for Exec_Dir use "exec";
   for Source_Files use ("PrintPopPush.adb", "Stack.adb", "Stack.ads",
"stackprodcons_io.adb");

   package Ide is
      for Documentation_Dir use "doc";
   end Ide;

   package Prove is
      for Switches use ("--level=2");
   end Prove;

   package Compiler is
      for Switches ("ada") use ("-gnat12", "-gnatw.y", "-gnatf");
   end Compiler;

   package Binder is
      for Switches ("ada") use ("-r");
   end Binder;

end Stack_Prod_Cons;
```

Stack.ads

```
-- SPARK 2014
pragma SPARK_Mode;
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);


package Stack
is

   Stack_Size : constant := 5;
   type Pointer_Stack is range 0 .. Stack_Size;
   subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
   type Vector is array (Index_Stack) of Integer;
   Size_ini   : Integer := 0;



   protected type StackPC is
      function Empty return Boolean
        ;
      function Full return Boolean
        ;
      procedure Push (I : in Integer)
        ;
      procedure Pop (I : out Integer)
        ;
   private
      Size_of_Stack : Pointer_Stack := 0;
      Data_in_Stack : Vector := Vector'(Index_Stack => 0);
      Not_Full      : Boolean := True;
   end StackPC;


   StackTask : StackPC;
```

```
   task type PushStack
     with
       Global => (In_Out => Stacktask),
     Depends  => (PushStack => PushStack,
                  StackTask => StackTask)
      ;
   task PopStack
     with
       Global => (In_Out => Stacktask),
     Depends  => (PopStack  => PopStack,
                  StackTask => StackTask)
    ;

   task PushStack1
     with
       Global => (In_Out => Stacktask),
     Depends  => (PushStack1 => PushStack1,
                  StackTask  => StackTask)
      ;

end Stack;
```

Stack.adb.

```
-- SPARK 2014
pragma SPARK_Mode;
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);

with Gnat.Io; use Gnat.Io;
with PrintPopPush;

package body Stack
is
   protected body StackPC
   is
      function Full return Boolean is (Size_of_Stack = Stack_Size);

      function Empty return Boolean is (Size_of_Stack = 0);

      procedure Push (I : in Integer)
      is
      begin
         Size_of_Stack := Size_of_Stack + 1;
         Data_in_stack (Size_of_stack) := I;
         if I >= 5000000 then
            PrintPopPush ("PUSH", "PushStack - Valor: " & Integer'Image (Data_in_stack
(Size_of_stack)) & " #########");
         else
            PrintPopPush ("PUSH", "PushStack - Valor: " & Integer'Image (Data_in_stack
(Size_of_stack)) & " <><><><><>");
         end if;
         pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para efeitos
de teste");
      end Push;

      procedure Pop (I : out Integer)
      is
      begin
         I := Data_in_stack (Size_of_stack);
         Size_of_Stack := Size_of_Stack - 1;
         PrintPopPush ("POP", "PopStack - Valor: " & Integer'Image (I) & " ----------------
----------");
         pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para efeitos
de teste");
         PrintPopPush ("SIZE", "PopStack - SizeStack: " & Integer'Image (Integer
(Size_of_Stack)) & "");
```

```
         pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para efeitos
de teste");
      end Pop;

   end StackPC;

   task body PushStack
   is
      nr     : Integer := 0;
      T_Full : Boolean := False;
   begin
      loop
         T_Full := StackTask.Full;
         if  not T_Full then
            if nr < Integer'Last then
               nr := nr + 1;
            else
               nr := 1;
            end if;
            StackTask.Push (nr);
         end if;
      end loop;
   end PushStack;

   task body PopStack is
      nr      : Integer := 0;
      T_Empty : Boolean := False;
   begin
      loop
         T_Empty := StackTask.Empty;
         if not T_Empty then
            StackTask.Pop (nr);
         end if;

      end loop;
   end PopStack;

   task body PushStack1
   is
      nr     : Integer := 5000000;
      T_Full : Boolean := False;
   begin
      loop
         T_Full := StackTask.Full;
         if  not T_Full then
            if nr < Integer'Last then
               nr := nr + 1;
            else
               nr := 5000000;
            end if;
            StackTask.Push (nr);
         end if;
      end loop;
   end PushStack1;

end Stack;
```

Stackprodcons_Io.adb

```
-- SPARK 2014
pragma SPARK_Mode;
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);

with Stack; use Stack;
procedure Stackprodcons_Io is
begin
   null;
end Stackprodcons_Io;
```

PrintPopPush.adb

```
with Gnat.Io; use Gnat.Io;
with Stack; Use Stack;
procedure PrintPopPush (str : in String; str1 : in String)
is
begin
    if str = "POP" then
        Put_Line (str1);
    else
        if str = "SIZE" then
            Put_Line (str1);
        else
            if str = "PUSH" then
                Put_Line (str1);
            end if;
        end if;
    end if;

end PrintPopPush;
```

Output de Stackprodcons_Io

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese>D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod
_Cons_T
ese\exec\stackprodcons_io.exe
PushStack - Valor:  5000001 #########
PushStack - Valor:  5000002 #########
PushStack - Valor:  5000003 #########
PushStack - Valor:  5000004 #########
PushStack - Valor:  5000005 #########
PopStack - Valor:  5000005 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  5000004 ----------------------------
PopStack - SizeStack:  3
PopStack - Valor:  5000003 ----------------------------
PopStack - SizeStack:  2
PopStack - Valor:  5000002 ----------------------------
PopStack - SizeStack:  1
PopStack - Valor:  5000001 ----------------------------
PopStack - SizeStack:  0
PushStack - Valor:  5000006 #########
PushStack - Valor:  5000007 #########
PushStack - Valor:  5000008 #########
PushStack - Valor:  5000009 #########
PushStack - Valor:  5000010 #########
```

Stack Concorrente usando uma entry

Configuração projecto igual ao anterior

Stack.ads

```
-- SPARK 2014
pragma SPARK_Mode;
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);
```

```
package Stack
is

   Stack_Size : constant := 5;
   type Pointer_Stack is range 0 .. Stack_Size;
   subtype Index_Stack is Pointer_Stack range 1 .. Stack_Size;
   type Vector is array (Index_Stack) of Integer;
   Size_ini   : Integer := 0;



   protected type StackPC is
      function Empty return Boolean
        ;
      function Full return Boolean
        ;
      entry Push (I : in Integer)
        ;
      procedure Pop (I : out Integer)
        ;
   private
      Size_of_Stack : Pointer_Stack := 0;
      Data_in_Stack : Vector := Vector'(Index_Stack => 0);
      Not_Full      : Boolean := True;
   end StackPC;


   StackTask : StackPC;

   task type PushStack
     with
       Global => (In_Out => Stacktask),
     Depends  => (PushStack => PushStack,
                  StackTask => StackTask)
       ;
   task PopStack
     with
       Global => (In_Out => Stacktask),
     Depends  => (PopStack  => PopStack,
                  StackTask => StackTask)
    ;

   task PushStack1
     with
       Global => (In_Out => Stacktask),
     Depends  => (PushStack1 => PushStack1,
                  StackTask  => StackTask)
      ;

end Stack;
```

Stack.adb

```
-- SPARK 2014
pragma SPARK_Mode;
pragma Profile (Ravenscar);
pragma Partition_Elaboration_Policy (Sequential);

with Gnat.Io; use Gnat.Io;
with PrintPopPush;

package body Stack1
is

    protected body StackPC
    is
        function Full return Boolean is (Size_of_Stack = Stack_Size);

        function Empty return Boolean is (Size_of_Stack = 0);
```

```
        entry Push (I : in Integer) when Not_Full
        is
        begin
            Not_Full := false;
            if not Full then
                Size_of_Stack := Size_of_Stack + 1;
                Data_in_stack (Size_of_stack) := I;
                PrintPopPush ("PUSH", "PushStack - Valor: " & Integer'Image (Data_in_stack
(Size_of_stack)) & " <><><><><>");
                pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para
efeitos de teste");
            end if;
            Not_Full := True;
        end Push;

        procedure Pop (I : out Integer)
        is
        begin
            I := 0;
            if not Empty then
                I := Data_in_stack (Size_of_stack);
                Size_of_Stack := Size_of_Stack - 1;
                PrintPopPush ("POP", "PopStack - Valor: " & Integer'Image (I) & " ----------
----------------");
                pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para
efeitos de teste");
                PrintPopPush ("SIZE", "PopStack - SizeStack: " & Integer'Image (Integer
(Size_of_Stack)) & "");
                pragma Annotate (GNATprove, Intentional, "", "PrintPopPush so' existe para
efeitos de teste");
            end if;
            Not_Full := true;
        end Pop;

    end StackPC;

    task body PushStack
    is
        nr     : Integer := 0;
        T_Full : Boolean := False;
    begin
        loop
            if nr < Integer'Last then
                nr := nr + 1;
            else
                nr := 1;
            end if;
            StackTask.Push (nr);
        end loop;
    end PushStack;

    task body PopStack is
        nr    : Integer := 0;
    begin
        loop
            StackTask.Pop (nr);
        end loop;
    end PopStack;

end Stack1;
```

Stackprodcons_Io.adb e PrintPopPush.adb igual ao projeto anterior

Output de Stackprodcons_Io

```
D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Prod_Cons_Tese1>D:\_DISSERTACAO\_EXEMPLOS_TESE\Stack_Pro
d_Cons_
Tese1\exec\stackprodcons_io.exe
PushStack - Valor:  1 <><><><><>
PushStack - Valor:  2 <><><><><>
PushStack - Valor:  3 <><><><><>
PushStack - Valor:  4 <><><><><>
PushStack - Valor:  5 <><><><><>
PopStack - Valor:  5 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  4 ----------------------------
PopStack - SizeStack:  3
PopStack - Valor:  3 ----------------------------
PopStack - SizeStack:  2
PopStack - Valor:  2 ----------------------------
PopStack - SizeStack:  1
PopStack - Valor:  1 ----------------------------
PopStack - SizeStack:  0
PushStack - Valor:  12 <><><><><>
PushStack - Valor:  13 <><><><><>
PushStack - Valor:  14 <><><><><>
PushStack - Valor:  15 <><><><><>
PushStack - Valor:  16 <><><><><>
PopStack - Valor:  16 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  15 ----------------------------
PopStack - SizeStack:  3
PopStack - Valor:  14 ----------------------------
PopStack - SizeStack:  2
PopStack - Valor:  13 ----------------------------
PopStack - SizeStack:  1
PopStack - Valor:  12 ----------------------------
PopStack - SizeStack:  0
PushStack - Valor:  51 <><><><><>
PushStack - Valor:  52 <><><><><>
PushStack - Valor:  53 <><><><><>
PushStack - Valor:  54 <><><><><>
PushStack - Valor:  55 <><><><><>
PopStack - Valor:  55 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  54 ----------------------------
PopStack - SizeStack:  3
PopStack - Valor:  53 ----------------------------
PopStack - SizeStack:  2
PopStack - Valor:  52 ----------------------------
PopStack - SizeStack:  1
PopStack - Valor:  51 ----------------------------
PopStack - SizeStack:  0
PushStack - Valor:  68 <><><><><>
PushStack - Valor:  69 <><><><><>
PushStack - Valor:  70 <><><><><>
PushStack - Valor:  71 <><><><><>
PushStack - Valor:  72 <><><><><>
PopStack - Valor:  72 ----------------------------
PopStack - SizeStack:  4
PopStack - Valor:  71 ----------------------------
PopStack - SizeStack:  3
```