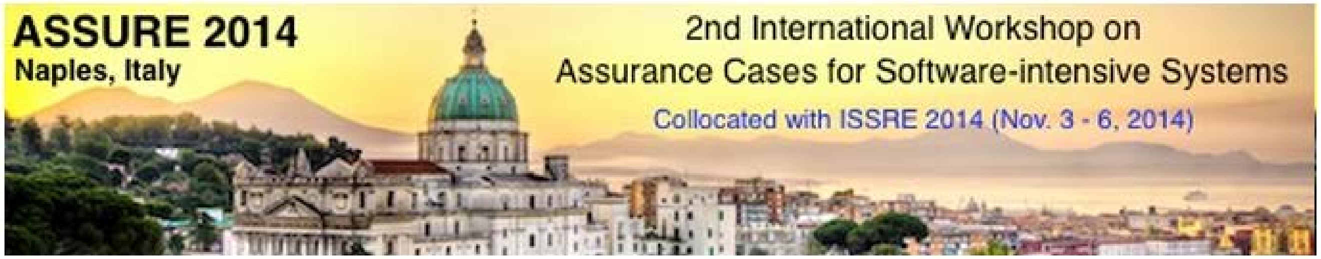

ASSURE 2014
Naples, Italy
2nd International Workshop on
Assurance Cases for Software-intensive Systems
Collocated with ISSRE 2014 (Nov. 3 - 6, 2014)

A Technique for Demonstrating Safety and Correctness of Program Translators : Strategy and Case Study

Eui-Sub Kim , Junbeom Yoo
Dependable Software Laboratory
KONKUK University, Republic of Korea

Jong-Gyun Choi , Young Jun Lee , Jang-Soo Lee
Man-Machine Interface System Laboratory
Korea Atomic Energy Research Institute, Republic of Korea

2014-11-05

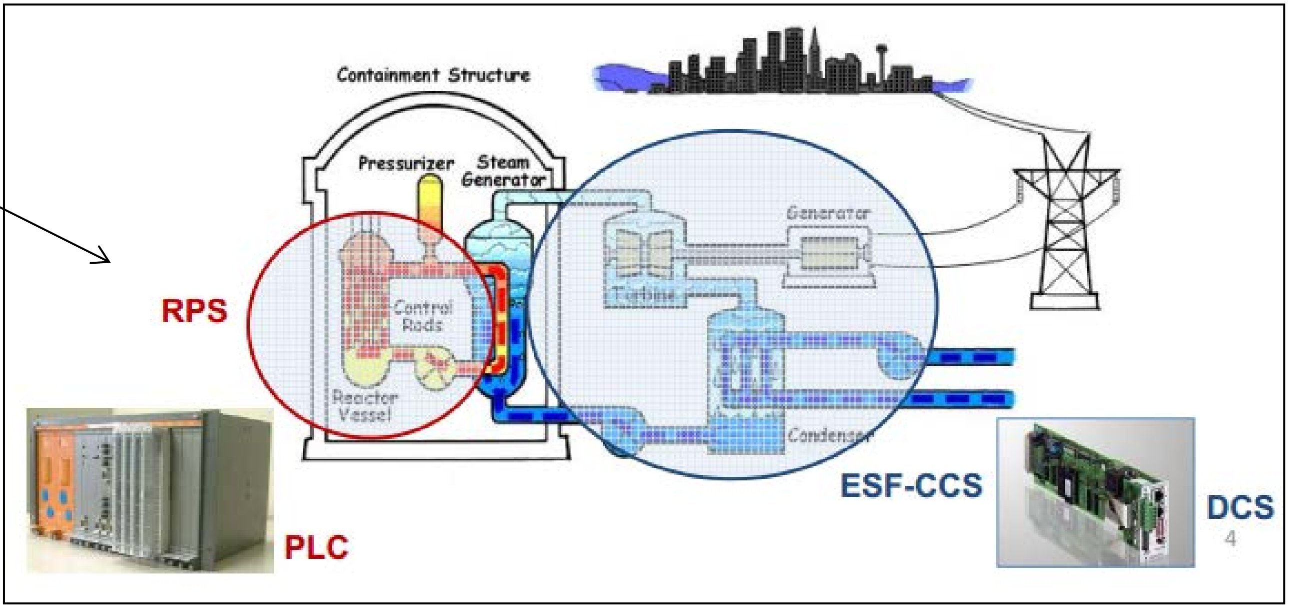
Contents

1. INTRODUCTION

1. Introduction

- Strategy and Case Study
  - **A specific translator : FBDtoVerilog**
  - **Case Study : BP** (Bistable Process) of RPS (Reactor Protection system) in Nuclear Power Plants
    - It produce the **'Shutdown' signal** to protect a NPP from unwanted situation.

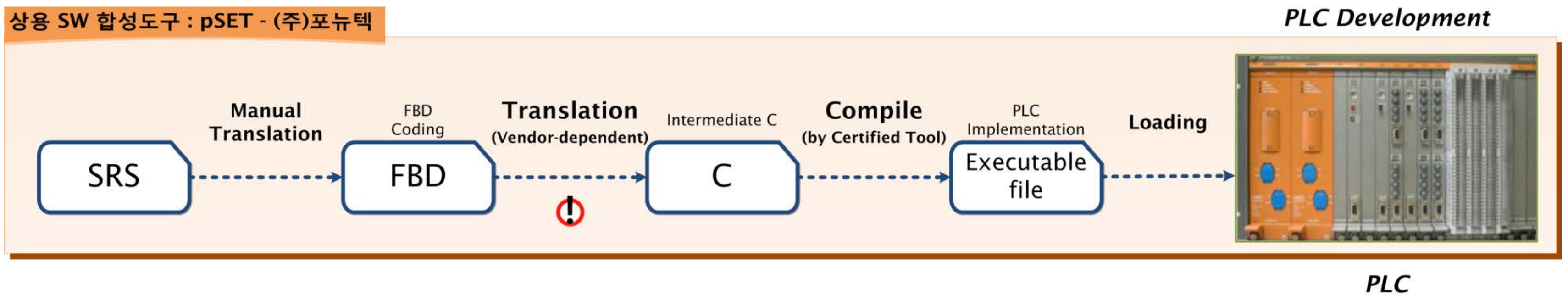
Scope



An overview of Nuclear Power Plants.

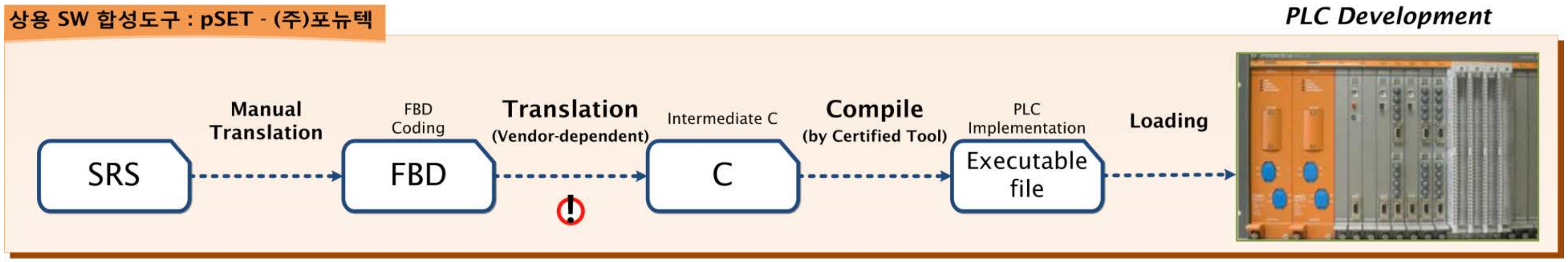
1. Introduction

- Software Development Process based on PLC



1. Introduction

- Software Development Process based on PLC



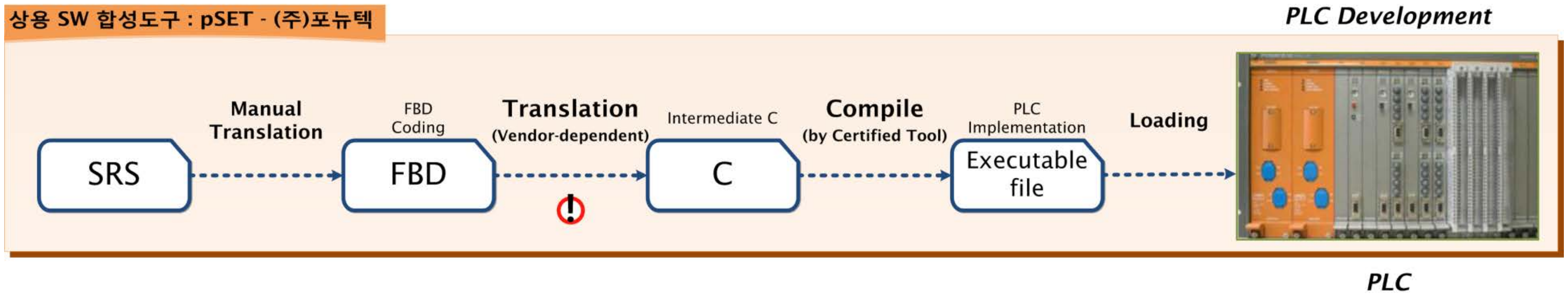
PLC

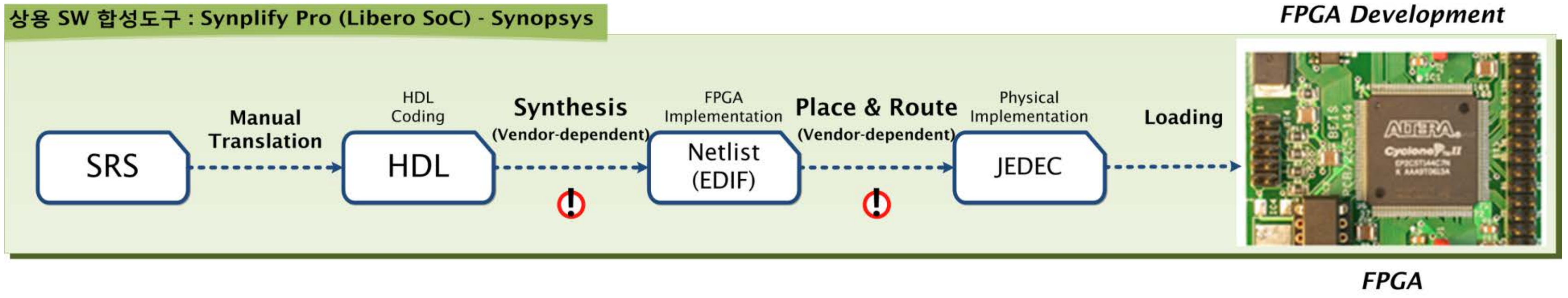
FPGA

Recently, there are trend
to replace the platform from PLC to FPGA

1. Introduction

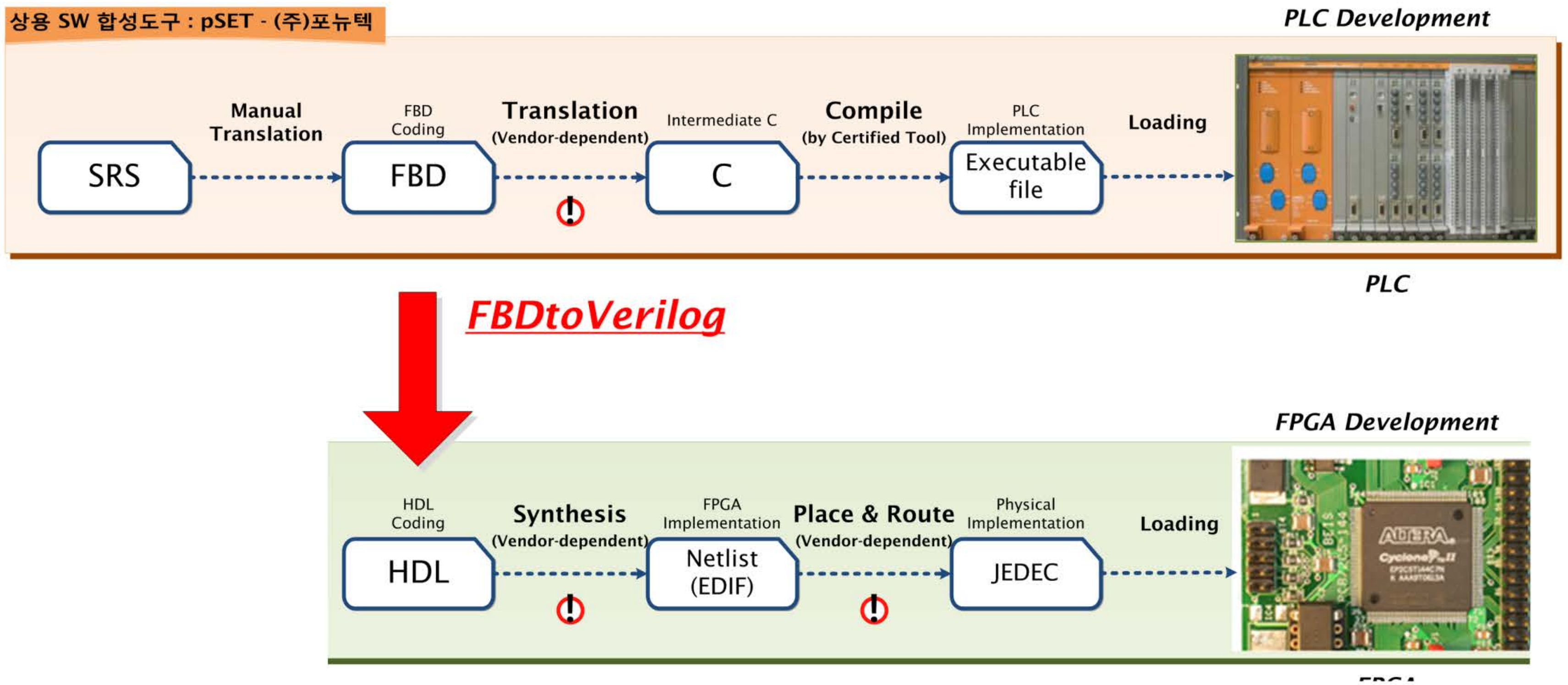
- PLC vs. FPGA
  - There have differences in stage of software development process





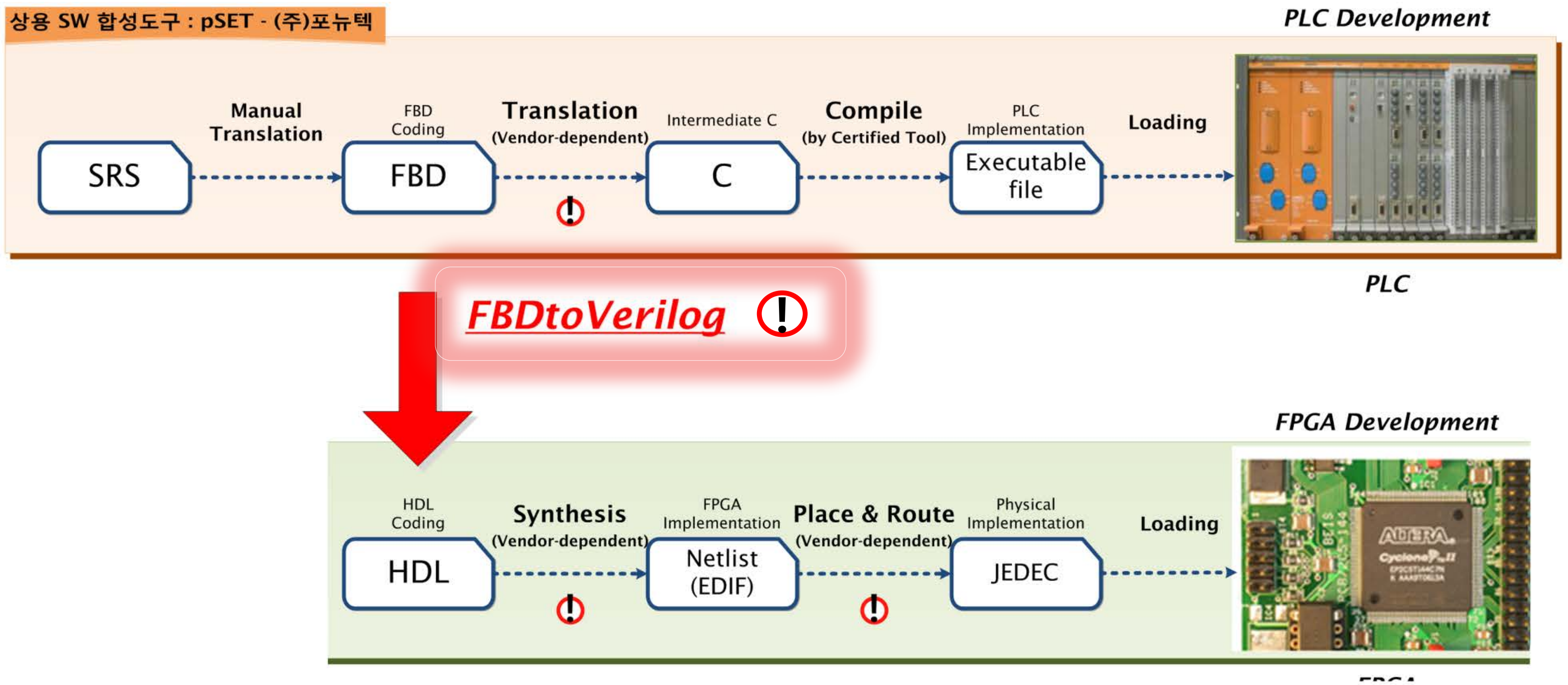
1. Introduction

- We developed the **FBDtoVerilog** translator
  - It **automatically** translates an FBD to a Verilog program



1. Introduction

- We developed the **FBDtoVerilog** translator
  - It **automatically** translates an FBD to a Verilog program

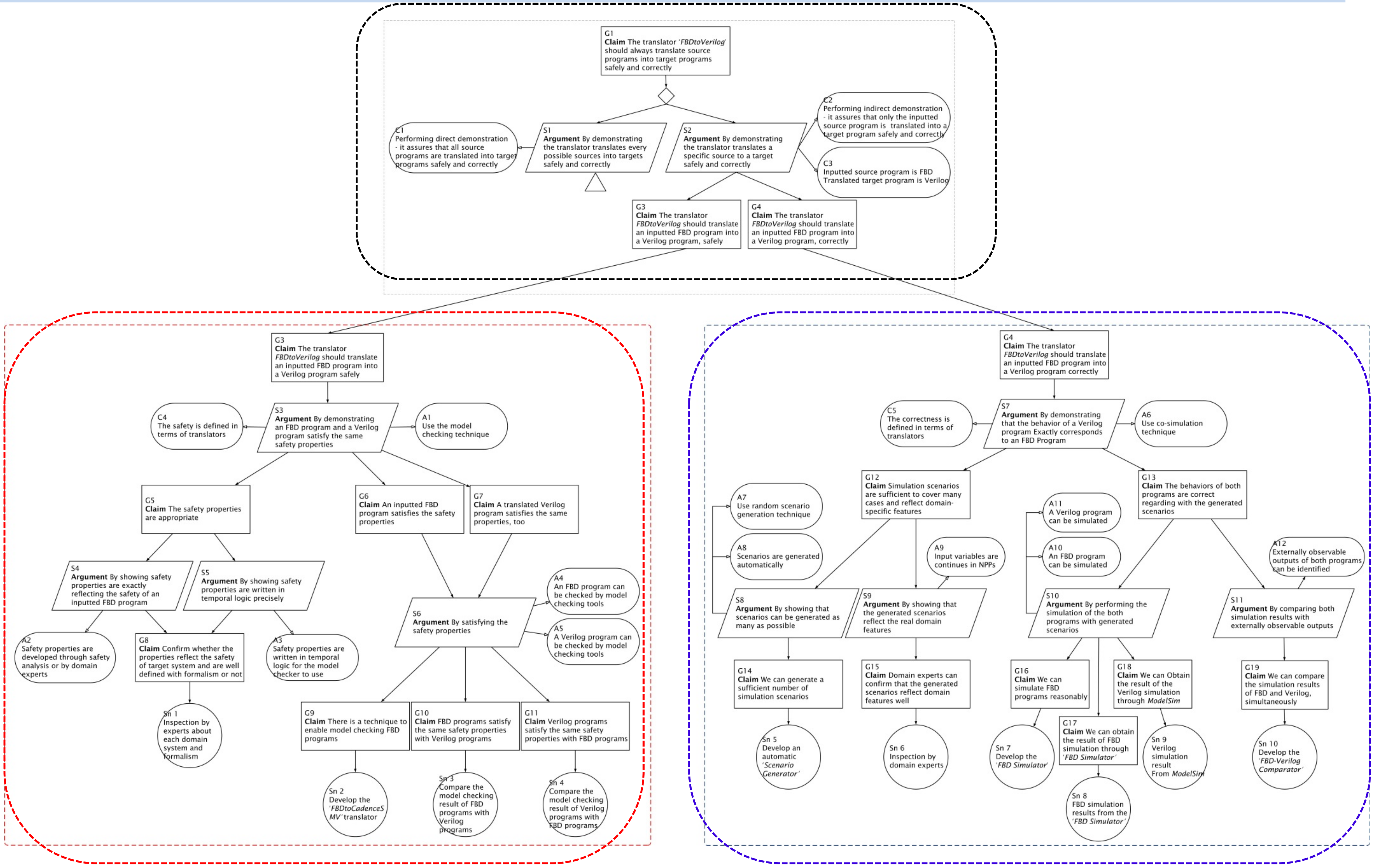


We must prove that the translator will work out correctly and safely

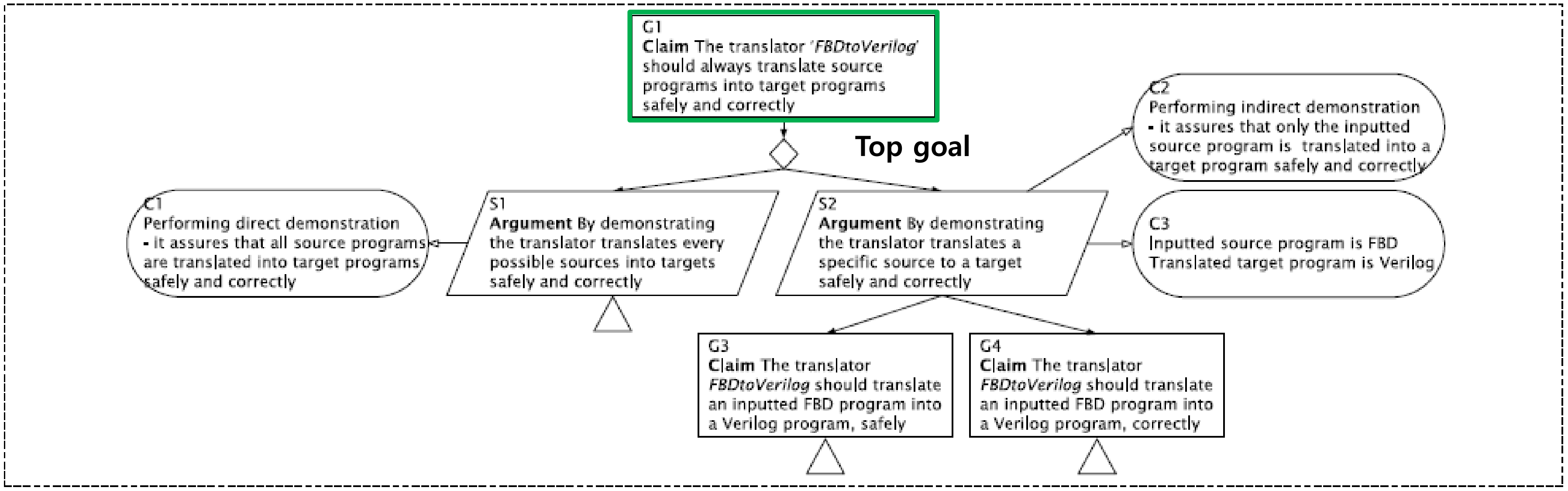
1. Safety Demonstration Strategy
2. Correctness Demonstration Strategy

2. A DEMONSTRATION STRATEGY

2. A Demonstration Strategy

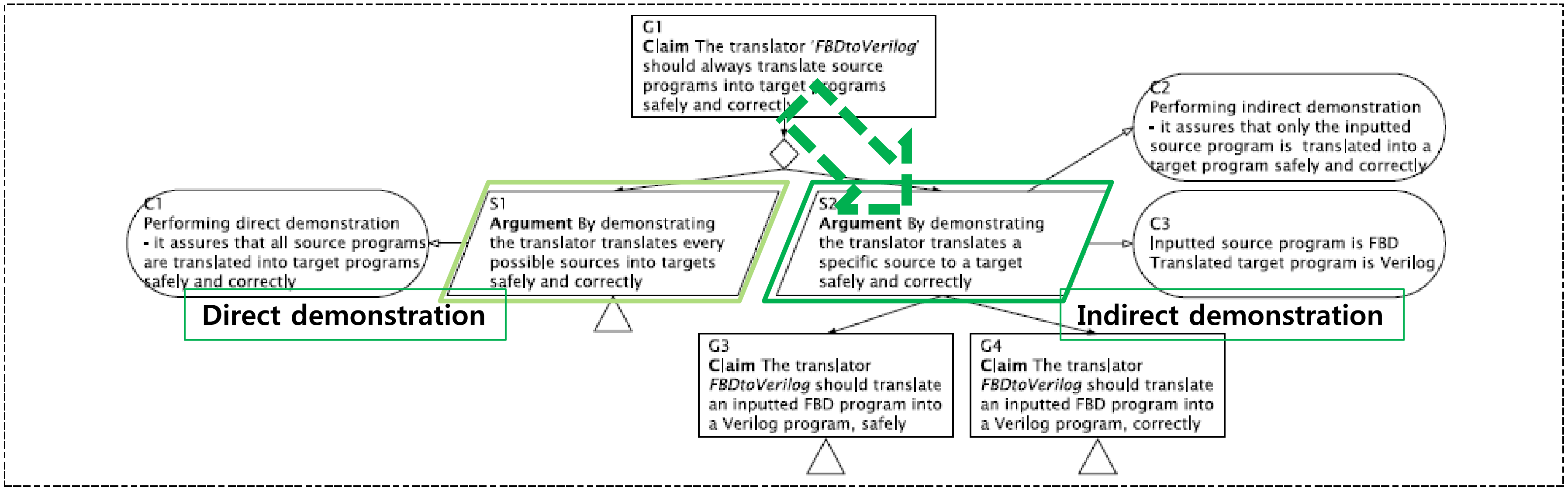

G1 Claim The translator 'FBDtoVerilog' should always translate source programs into target programs safely and correctly
C1 Performing direct demonstration - it assures that all source programs are translated into target programs safely and correctly
S1 Argument By demonstrating the translator translates every possible sources into targets safely and correctly
S2 Argument By demonstrating the translator translates a specific source to a target safely and correctly
C2 Performing indirect demonstration - it assures that only the inputted source program is translated into a target program safely and correctly
C3 Inputted source program is FBD Translated target program is Verilog
G3 Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, safely
G4 Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, correctly
G3 Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program safely
C4 The safety is defined in terms of translators
S3 Argument By demonstrating an FBD program and a Verilog program satisfy the same safety properties
A1 Use the model checking technique
G5 Claim The safety properties are appropriate
G6 Claim An inputted FBD program satisfies the safety properties
G7 Claim A translated Verilog program satisfies the same safety properties, too
S4 Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5 Argument By showing safety properties are written in temporal logic precisely
A4 An FBD program can be checked by model checking tools
A5 A Verilog program can be checked by model checking tools
S6 Argument By satisfying the safety properties
A2 Safety properties are developed through safety analysis or by domain experts
G8 Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3 Safety properties are written in temporal logic for the model checker to use
Sn 1 Inspection by experts about each domain system and formalism
G9 Claim There is a technique to enable model checking FBD programs
G10 Claim FBD programs satisfy the same safety properties with Verilog programs
G11 Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 2 Develop the 'FBDtoCadenceSMV' translator
Sn 3 Compare the model checking result of FBD programs with Verilog programs
Sn 4 Compare the model checking result of Verilog programs with FBD programs
G4 Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program correctly
C5 The correctness is defined in terms of translators
S7 Argument By demonstrating that the behavior of a Verilog program Exactly corresponds to an FBD Program
A6 Use co-simulation technique
A7 Use random scenario generation technique
A8 Scenarios are generated automatically
G12 Claim Simulation scenarios are sufficient to cover many cases and reflect domain-specific features
A9 Input variables are continues in NPPs
A11 A Verilog program can be simulated
A10 An FBD program can be simulated
G13 Claim The behaviors of both programs are correct regarding with the generated scenarios
A12 Externally observable outputs of both programs can be identified
S8 Argument By showing that scenarios can be generated as many as possible
S9 Argument By showing that the generated scenarios reflect the real domain features
S10 Argument By performing the simulation of the both programs with generated scenarios
S11 Argument By comparing both simulation results with externally observable outputs
G14 Claim We can generate a sufficient number of simulation scenarios
G15 Claim Domain experts can confirm that the generated scenarios reflect domain features well
G16 Claim We can simulate FBD programs reasonably
G18 Claim We can Obtain the result of the Verilog simulation through ModelSim
G19 Claim We can compare the simulation results of FBD and Verilog, simultaneously
G17 Claim We can obtain the result of FBD simulation through 'FBD Simulator'
Sn 5 Develop an automatic 'Scenario Generator'
Sn 6 Inspection by domain experts
Sn 7 Develop the 'FBD Simulator'
Sn 9 Verilog simulation result From ModelSim
Sn 10 Develop the 'FBD-Verilog Comparator'
Sn 8 FBD simulation results from the 'FBD Simulator'

2. A Demonstration Strategy


G1
Claim The translator 'FBDtoVerilog' should always translate source programs into target programs safely and correctly
Top goal
C2
Performing indirect demonstration
- it assures that only the inputted source program is translated into a target program safely and correctly
C1
Performing direct demonstration
- it assures that all source programs are translated into target programs safely and correctly
S1
Argument By demonstrating the translator translates every possible sources into targets safely and correctly
S2
Argument By demonstrating the translator translates a specific source to a target safely and correctly
C3
Inputted source program is FBD
Translated target program is Verilog
G3
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, safely
G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, correctly

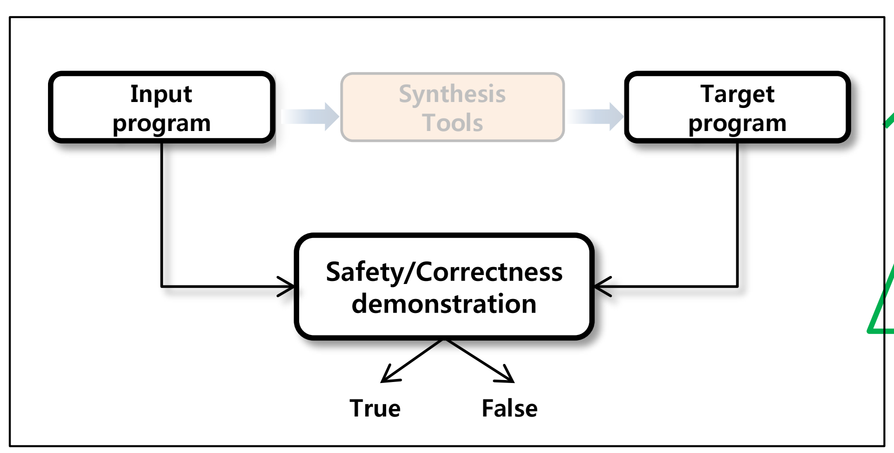
- Top goal : The translator 'FBDtoVerilog' should always translate source programs into target programs safely and correctly.

2. A Demonstration Strategy

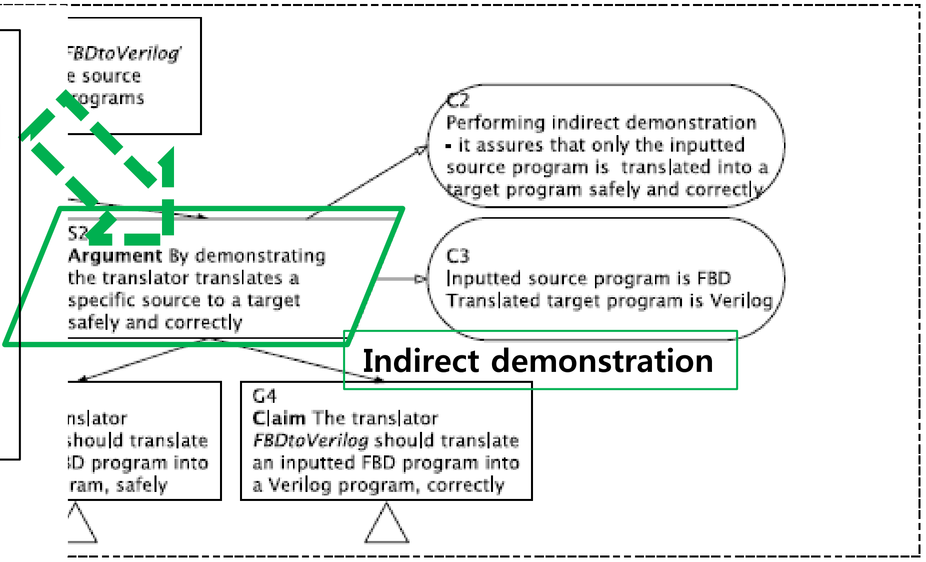

G1
Claim The translator 'FBDtoVerilog' should always translate source programs into target programs safely and correctly
C1
Performing direct demonstration
- it assures that all source programs are translated into target programs safely and correctly
S1
Argument By demonstrating the translator translates every possible sources into targets safely and correctly
S2
Argument By demonstrating the translator translates a specific source to a target safely and correctly
C2
Performing indirect demonstration
- it assures that only the inputted source program is translated into a target program safely and correctly
C3
Inputted source program is FBD
Translated target program is Verilog
Direct demonstration
Indirect demonstration
G3
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, safely
G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, correctly

- Direct demonstration approach
- Indirect demonstration approach

2. A Demonstration Strategy

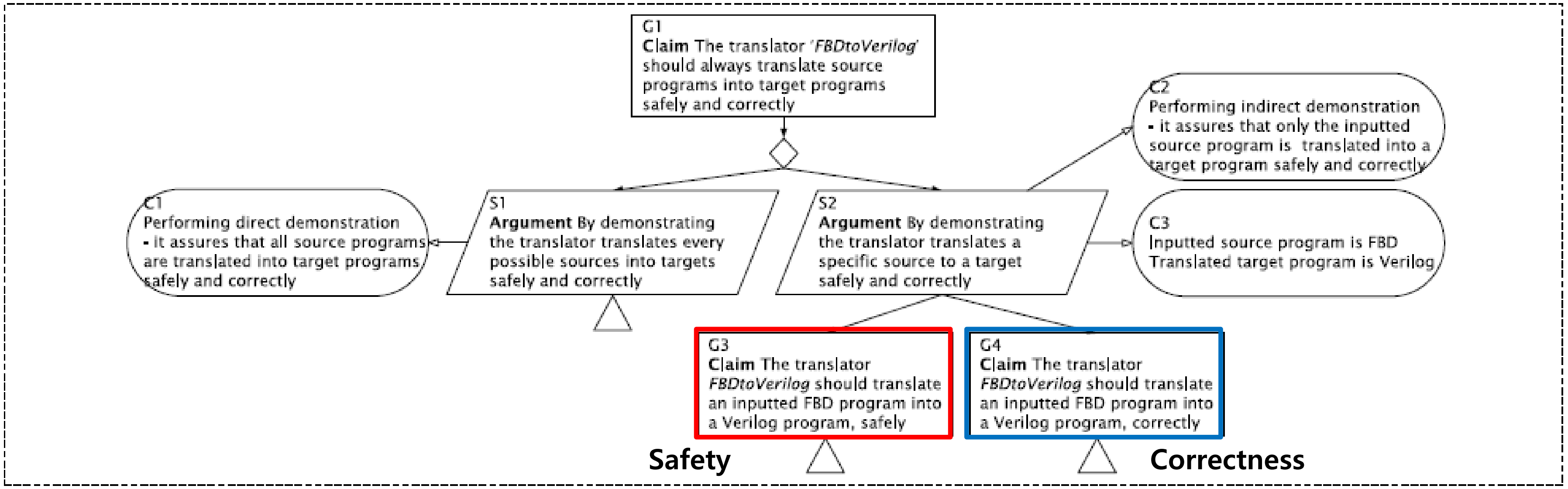


Indirect demonstration


FBDtoVerilog'
e source
ograms
C2
Performing indirect demonstration
- it assures that only the inputted source program is translated into a target program safely and correctly
S2
Argument By demonstrating the translator translates a specific source to a target safely and correctly
C3
Inputted source program is FBD
Translated target program is Verilog
Indirect demonstration
nslator
should translate
D program into
ram, safely
G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, correctly

- Direct demonstration approach
- **Indirect demonstration approach**

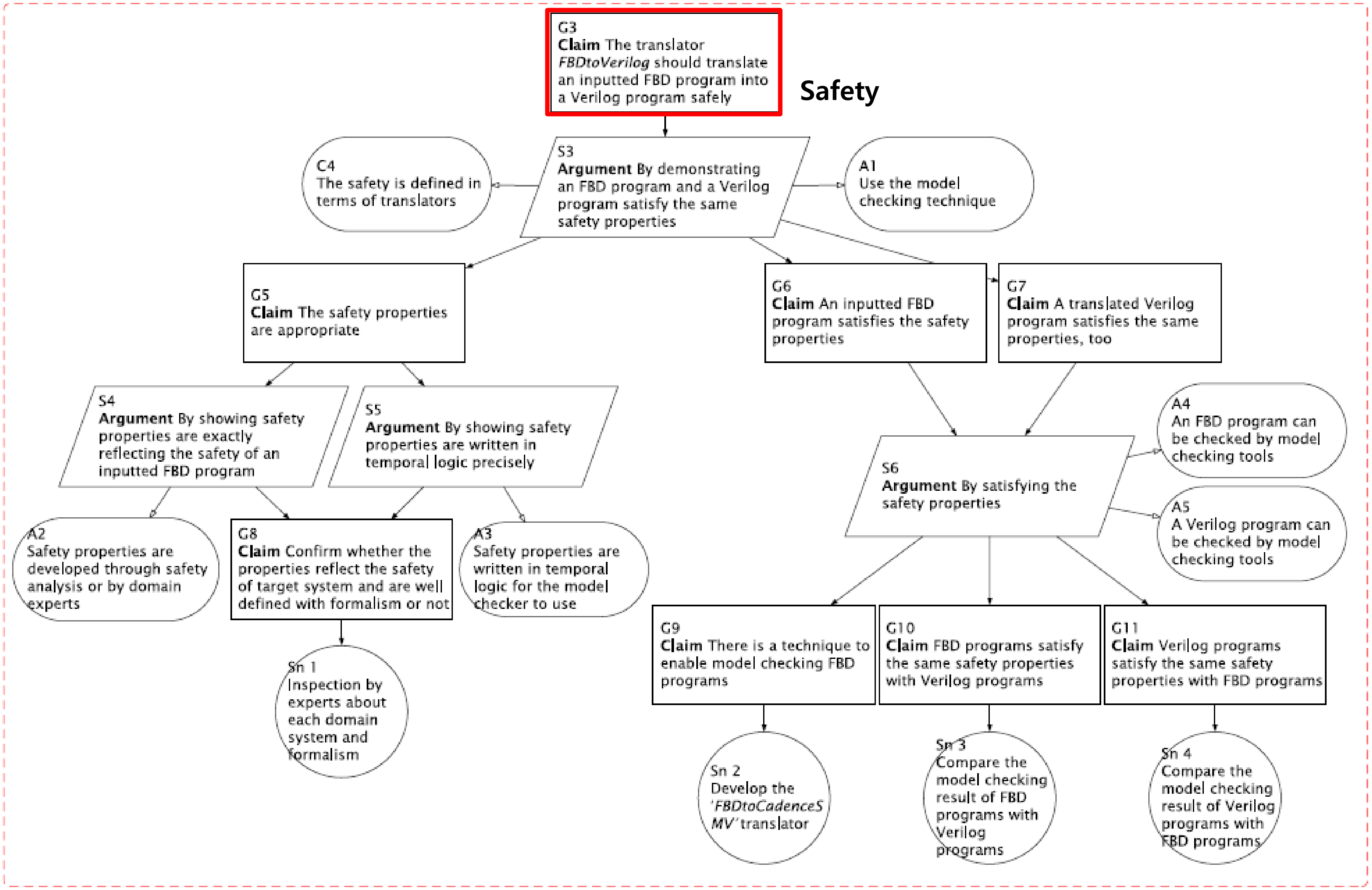
2. A Demonstration Strategy



G1
Claim The translator 'FBDtoVerilog' should always translate source programs into target programs safely and correctly
C1
Performing direct demonstration
- it assures that all source programs are translated into target programs safely and correctly
S1
Argument By demonstrating the translator translates every possible sources into targets safely and correctly
S2
Argument By demonstrating the translator translates a specific source to a target safely and correctly
C2
Performing indirect demonstration
- it assures that only the inputted source program is translated into a target program safely and correctly
C3
Inputted source program is FBD
Translated target program is Verilog
G3
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, safely
G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program, correctly
Safety
Correctness

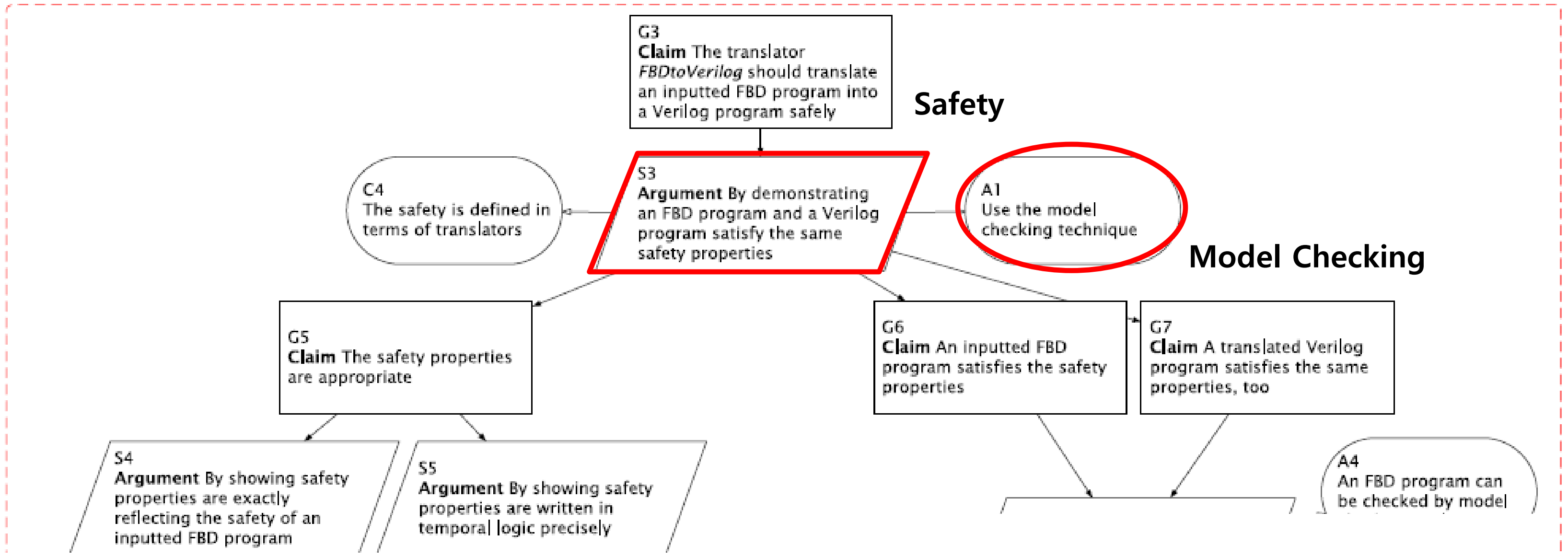
- Safety
  - Definition : A translator is safe, if safety properties are satisfied with the input and output programs simultaneously.
- Correctness
  - Definition : A translator is correct, if the behavior of a translated program is the same with its source program for all possible input scenarios.

2.1 The Safety Demonstration Strategy

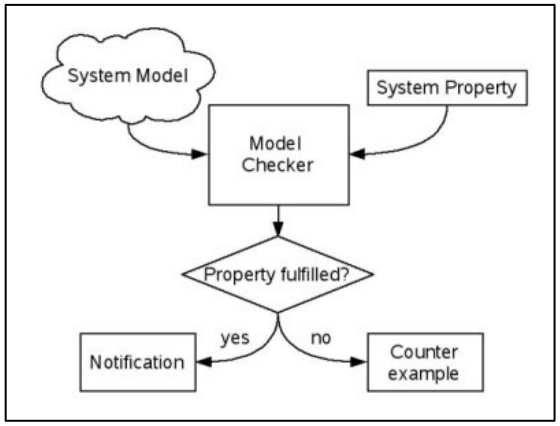


G3
Claim The translator *FBDtoVerilog* should translate an inputted FBD program into a Verilog program safely
Safety
S3
Argument By demonstrating an FBD program and a Verilog program satisfy the same safety properties
C4
The safety is defined in terms of translators
A1
Use the model checking technique
G5
Claim The safety properties are appropriate
G6
Claim An inputted FBD program satisfies the safety properties
G7
Claim A translated Verilog program satisfies the same properties, too
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
S6
Argument By satisfying the safety properties
A4
An FBD program can be checked by model checking tools
A5
A Verilog program can be checked by model checking tools
A2
Safety properties are developed through safety analysis or by domain experts
G8
Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3
Safety properties are written in temporal logic for the model checker to use
G9
Claim There is a technique to enable model checking FBD programs
G10
Claim FBD programs satisfy the same safety properties with Verilog programs
G11
Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 1
Inspection by experts about each domain system and formalism
Sn 2
Develop the '*FBDtoCadenceSMV*' translator
Sn 3
Compare the model checking result of FBD programs with Verilog programs
Sn 4
Compare the model checking result of Verilog programs with FBD programs

2.1 The Safety Demonstration Strategy

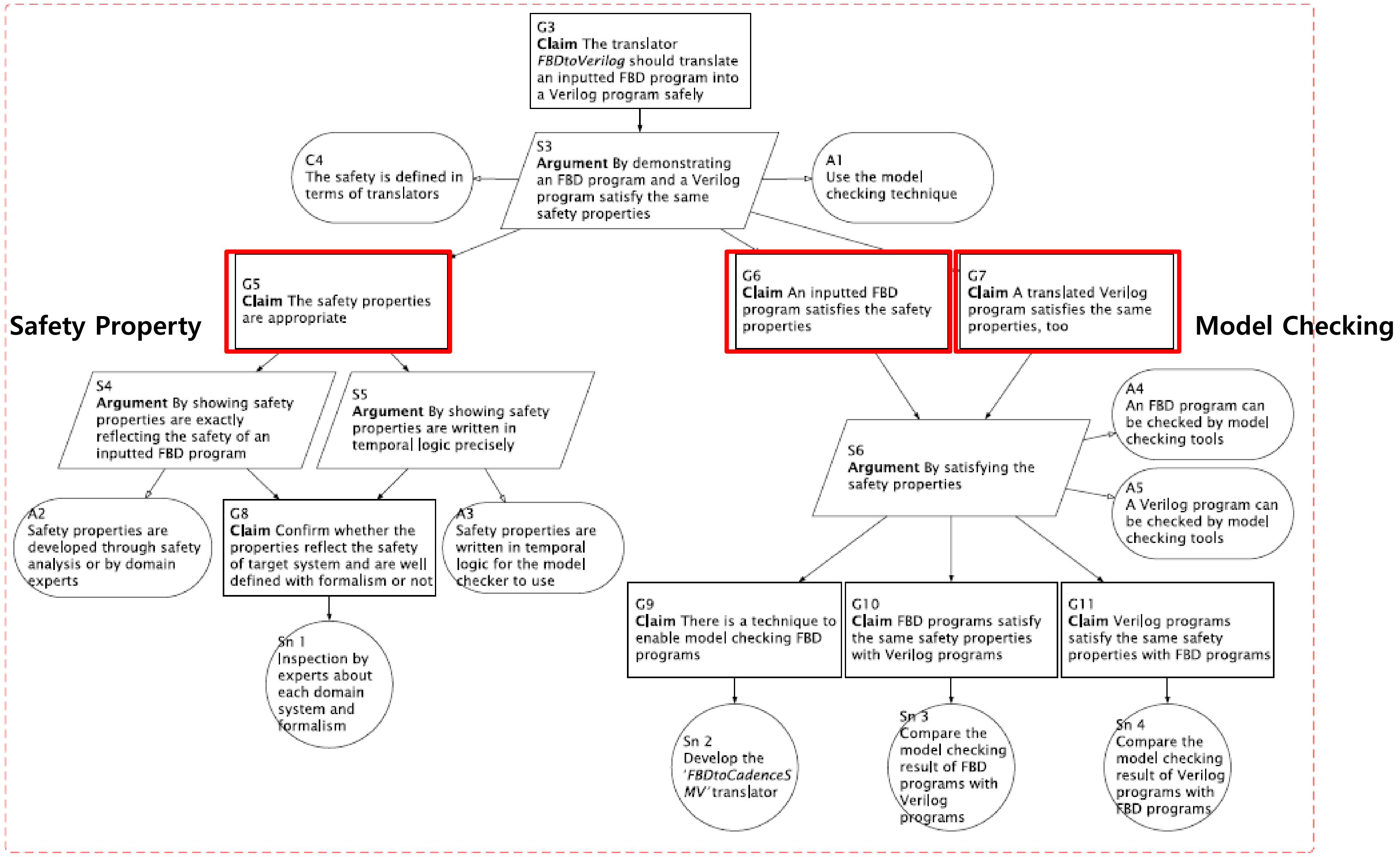

G3
Claim The translator *FBDtoVerilog* should translate an inputted FBD program into a Verilog program safely
Safety
C4
The safety is defined in terms of translators
S3
Argument By demonstrating an FBD program and a Verilog program satisfy the same safety properties
A1
Use the model checking technique
Model Checking
G5
Claim The safety properties are appropriate
G6
Claim An inputted FBD program satisfies the safety properties
G7
Claim A translated Verilog program satisfies the same properties, too
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
A4
An FBD program can be checked by model

- Model Checking
  - Given a model of a system, exhaustively and automatically check whether this model meets a given specification.
  - We used a model checking tool CadenceSMV

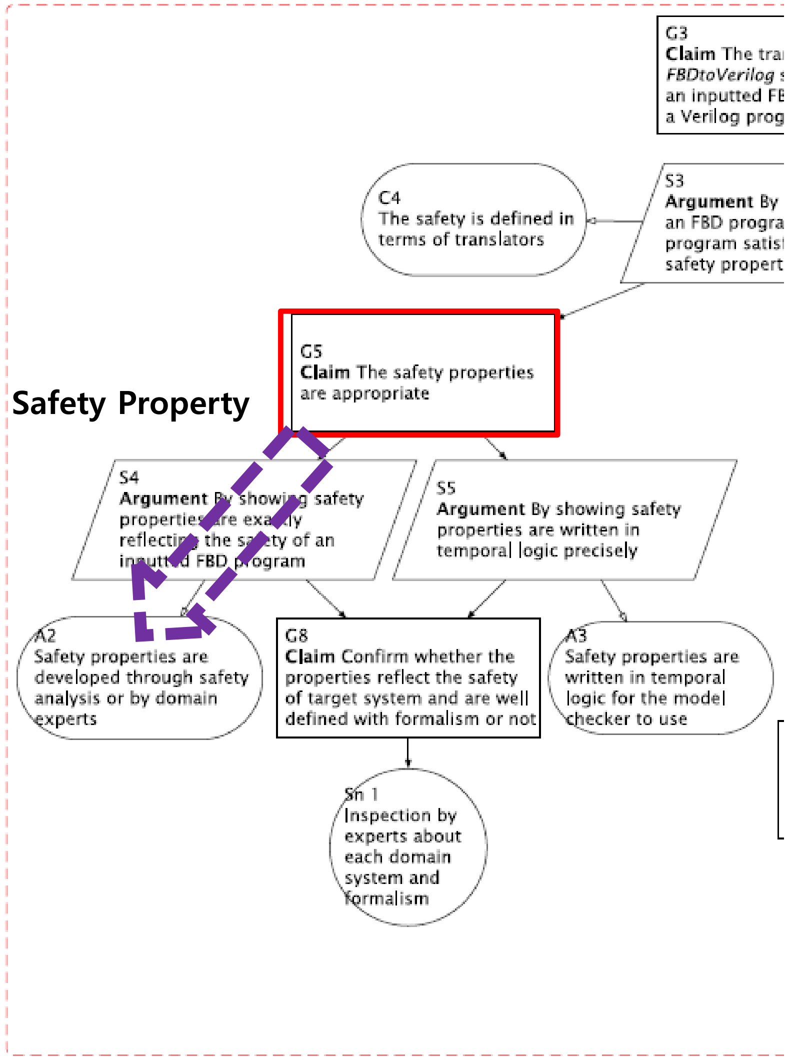


A typical model checking work-flow

2.1 The Safety Demonstration Strategy

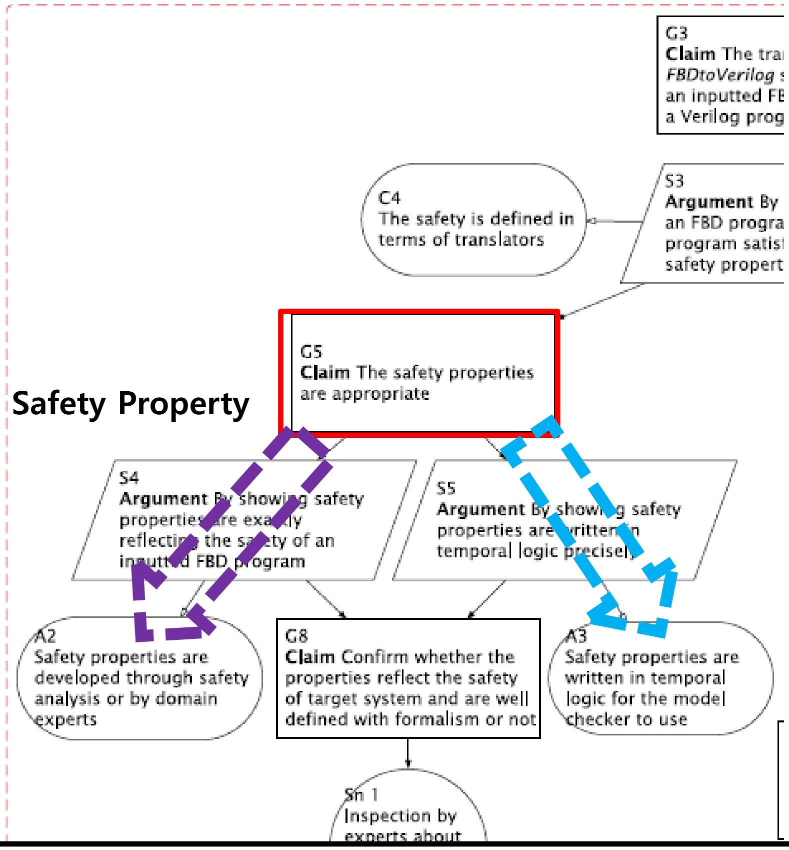

G3
Claim The translator *FBDtoVerilog* should translate an inputted FBD program into a Verilog program safely
C4
The safety is defined in terms of translators
S3
Argument By demonstrating an FBD program and a Verilog program satisfy the same safety properties
A1
Use the model checking technique
Safety Property
G5
Claim The safety properties are appropriate
G6
Claim An inputted FBD program satisfies the safety properties
G7
Claim A translated Verilog program satisfies the same properties, too
Model Checking
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
A4
An FBD program can be checked by model checking tools
S6
Argument By satisfying the safety properties
A5
A Verilog program can be checked by model checking tools
A2
Safety properties are developed through safety analysis or by domain experts
G8
Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3
Safety properties are written in temporal logic for the model checker to use
G9
Claim There is a technique to enable model checking FBD programs
G10
Claim FBD programs satisfy the same safety properties with Verilog programs
G11
Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 1
Inspection by experts about each domain system and formalism
Sn 2
Develop the '*FBDtoCadenceSMV*' translator
Sn 3
Compare the model checking result of FBD programs with Verilog programs
Sn 4
Compare the model checking result of Verilog programs with FBD programs

2.1 The Safety Demonstration Strategy


G3
Claim The tra
FBDtoVerilog
an inputted FE
a Verilog prog
C4
The safety is defined in terms of translators
S3
Argument By
an FBD progra
program satis
safety propert
Safety Property
G5
Claim The safety properties are appropriate
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
A2
Safety properties are developed through safety analysis or by domain experts
G8
Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3
Safety properties are written in temporal logic for the model checker to use
Sn 1
Inspection by experts about each domain system and formalism

- Goal 5
  - Claim the safety <u>**properties are appreciate**</u>
- Assumption 2
  - Safety properties are <u>**reflecting important safety features**</u> of the target input/output programs

2.1 The Safety Demonstration Strategy


G3
Claim The tra
FBDtoVerilog
an inputted FE
a Verilog prog
C4
The safety is defined in terms of translators
S3
Argument By
an FBD progra
program satis
safety propert
Safety Property
G5
Claim The safety properties are appropriate
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
A2
Safety properties are developed through safety analysis or by domain experts
G8
Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3
Safety properties are written in temporal logic for the model checker to use
Sn 1
Inspection by experts about

- Goal 5
  - Claim the safety **properties are appreciate**
- Assumption 2
  - Safety properties are **reflecting important safety features** of the target input/output programs
- Assumption 3
  - They **well formed with the formalism** which the model checking technique requires

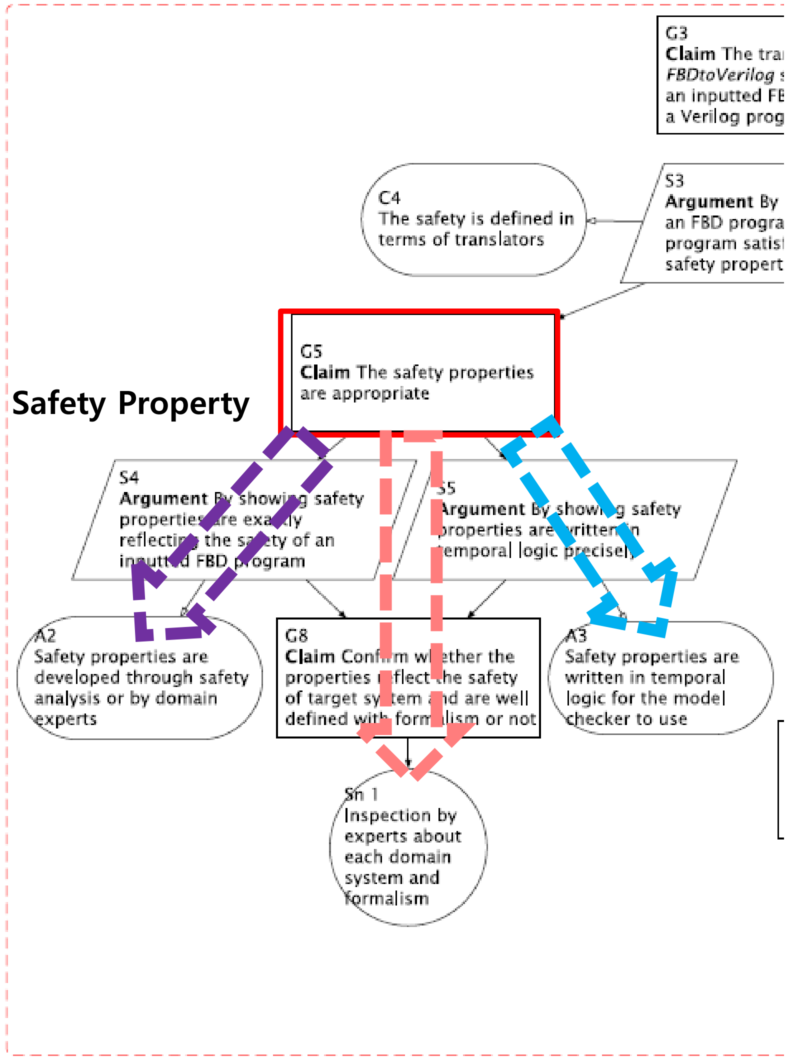
Natural requirement

"If PV_OUT (An input sensor value) is more than the TSP (Trip Set-Point) for a predefined time, then the trip signal should be fired (TRIP_LOGIC = 1) immediately."

CTL formula

: AG((PV_OUT > TSP) & (TRIP_CNT >= (MAXCNT - 1)) → AX(TRIP_LOGIC = 1))

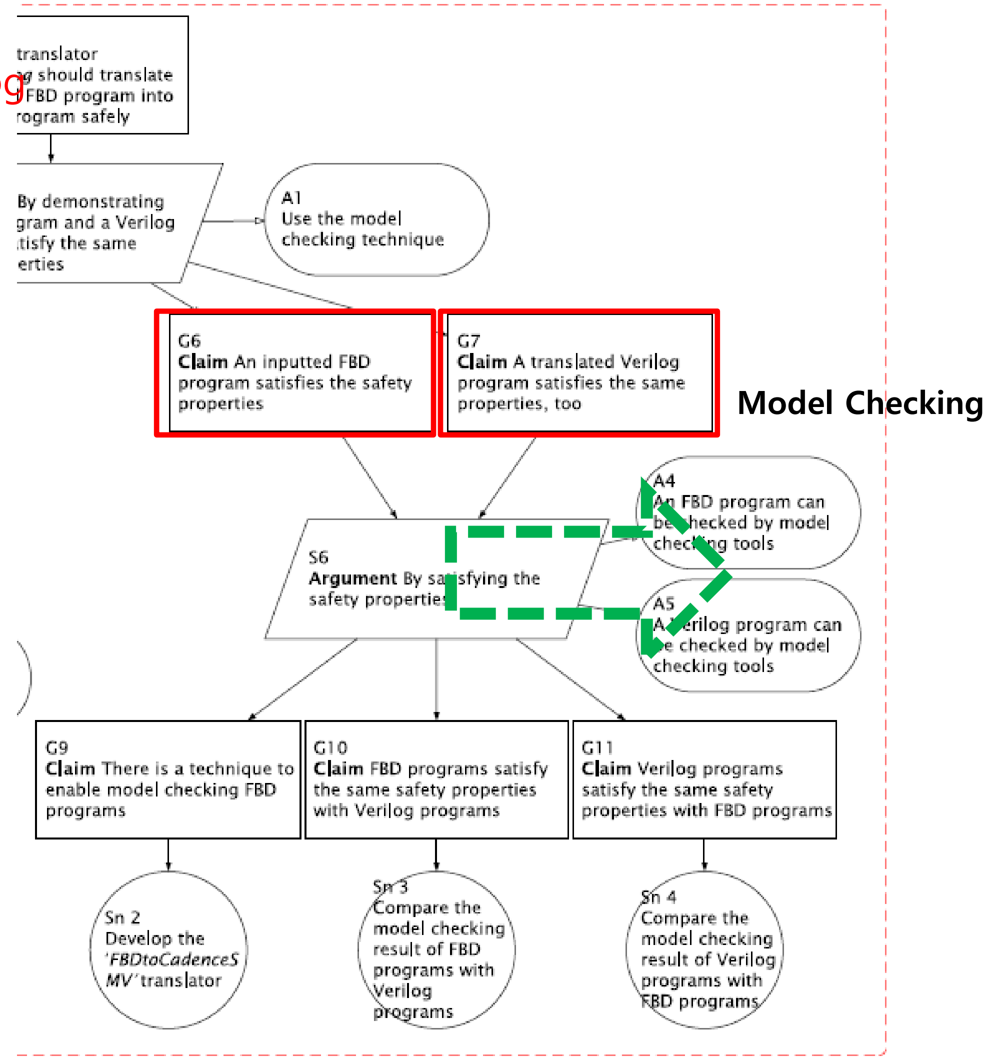
2.1 The Safety Demonstration Strategy


G3
Claim The tra
FBDtoVerilog
an inputted FE
a Verilog prog
C4
The safety is defined in terms of translators
S3
Argument By
an FBD progra
program satis
safety propert
G5
Claim The safety properties are appropriate
Safety Property
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
A2
Safety properties are developed through safety analysis or by domain experts
G8
Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3
Safety properties are written in temporal logic for the model checker to use
Sn 1
Inspection by experts about each domain system and formalism

- Goal 5
  - Claim the safety **properties are appreciate**
- Assumption 2
  - Safety properties are **reflecting important safety features** of the target input/output programs
- Assumption 3
  - They **well formed with the formalism** which the model checking technique requires
- Evidence
  - **Inspection by experts** about each domain system and formalism

2.1 The Safety Demonstration Strategy

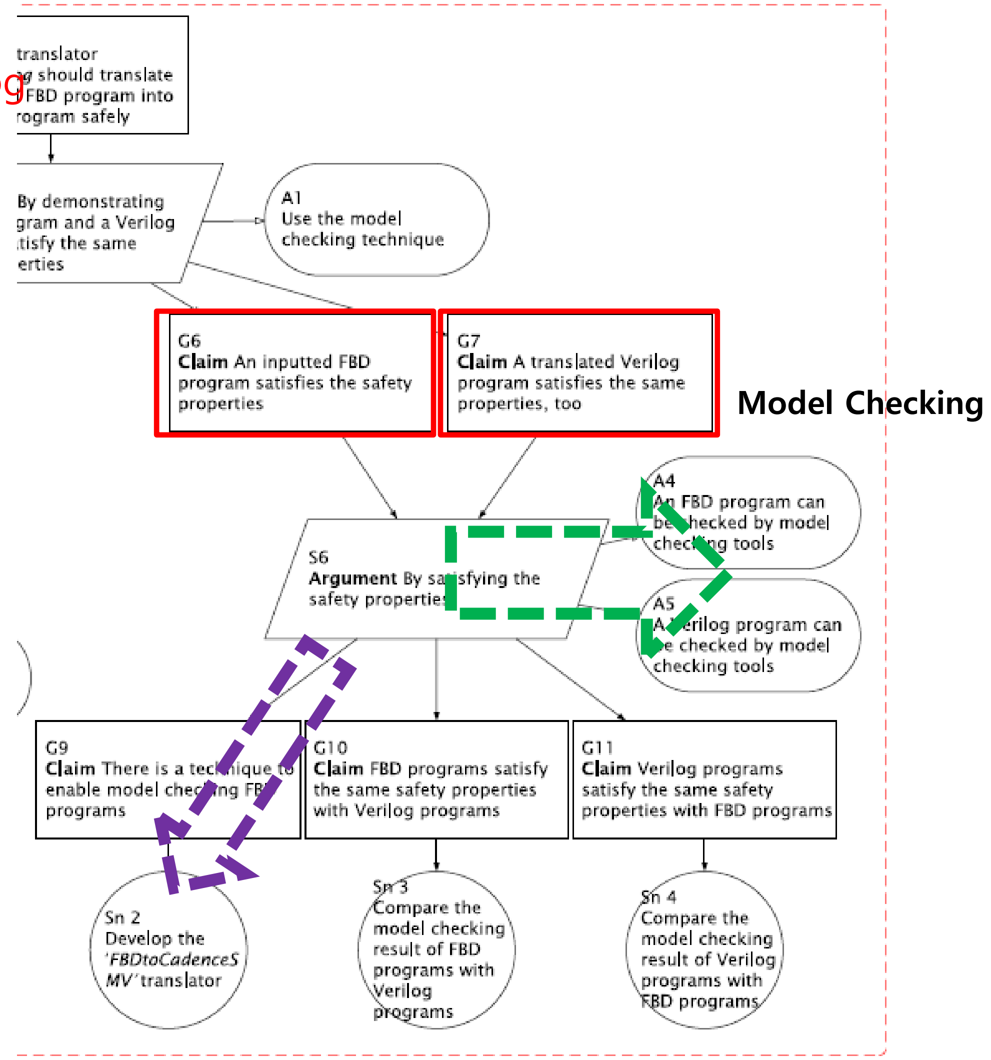
- Goal 6, 7
  - An Inputted FBD and a translated Verilog <u>satisfy the same porpoises</u>
- Assumption 4, 5
  - Are there model checker for FBD and Verilog?



translator
g should translate FBD program into rogram safely
By demonstrating gram and a Verilog tisfy the same erties
A1 Use the model checking technique
G6 Claim An inputted FBD program satisfies the safety properties
G7 Claim A translated Verilog program satisfies the same properties, too
Model Checking
A4 An FBD program can be checked by model checking tools
S6 Argument By satisfying the safety properties
A5 A Verilog program can be checked by model checking tools
G9 Claim There is a technique to enable model checking FBD programs
G10 Claim FBD programs satisfy the same safety properties with Verilog programs
G11 Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 2 Develop the 'FBDtoCadenceSMV' translator
Sn 3 Compare the model checking result of FBD programs with Verilog programs
Sn 4 Compare the model checking result of Verilog programs with FBD programs

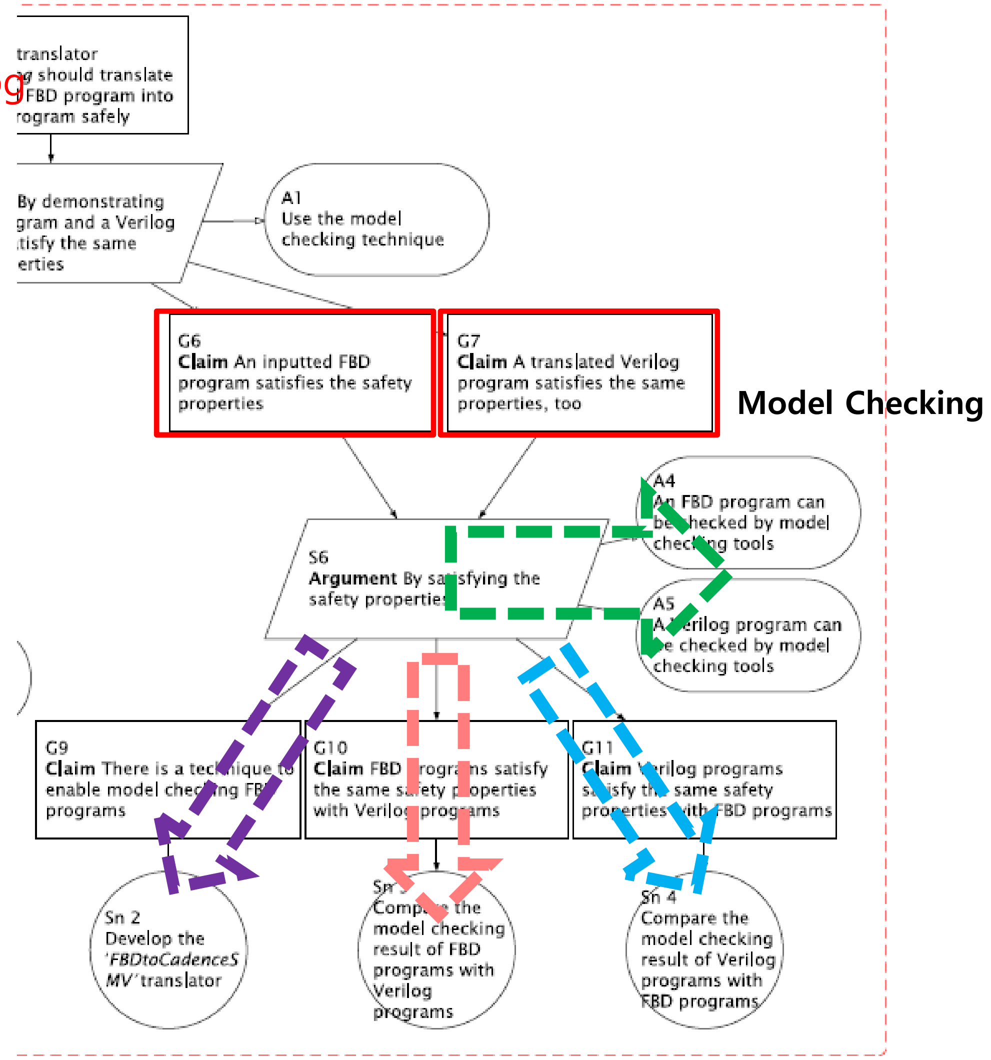
2.1 The Safety Demonstration Strategy

- Goal 6, 7
  - An Inputted FBD and a translated Verilog <u>satisfy the same porpoises</u>
- Assumption 4, 5
  - Are there model checker for FBD and Verilog?
- Evidence 2
  - Claim that there are model checkers to check both programs
    → We developed the *'FBDtoCadenceSMV*


translator
g should translate
FBD program into
rogram safely
By demonstrating
gram and a Verilog
tisfy the same
erties
A1
Use the model checking technique
G6
Claim An inputted FBD program satisfies the safety properties
G7
Claim A translated Verilog program satisfies the same properties, too
Model Checking
A4
An FBD program can be checked by model checking tools
A5
A Verilog program can be checked by model checking tools
S6
Argument By satisfying the safety properties
G9
Claim There is a technique to enable model checking FBD programs
G10
Claim FBD programs satisfy the same safety properties with Verilog programs
G11
Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 2
Develop the 'FBDtoCadenceSMV' translator
Sn 3
Compare the model checking result of FBD programs with Verilog programs
Sn 4
Compare the model checking result of Verilog programs with FBD programs

2.1 The Safety Demonstration Strategy

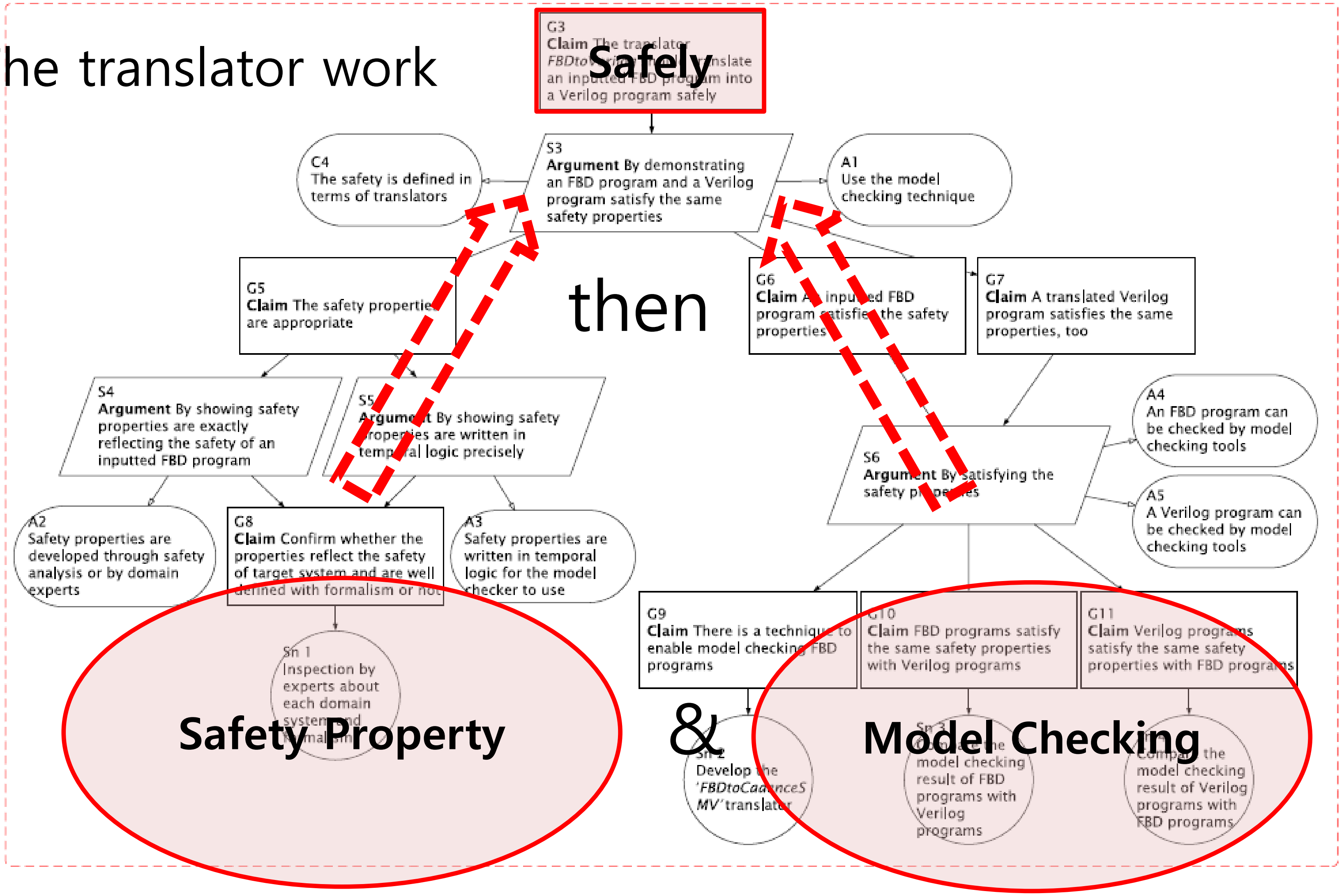
- Goal 6, 7
  - An Inputted FBD and a translated Verilog <u>satisfy the same porpoises</u>
- Assumption 4, 5
  - Are there model checker for FBD and Verilog?
- Evidence 2
  - Claim that there are model checkers to check both programs → We developed the *'FBDtoCadenceSMV'*
- Evidence 3
  - FBD model checking result
- Evidence 4
  - Verilog model checking result



translator
g should translate
FBD program into
rogram safely
By demonstrating
gram and a Verilog
tisfy the same
erties
A1
Use the model checking technique
G6
Claim An inputted FBD program satisfies the safety properties
G7
Claim A translated Verilog program satisfies the same properties, too
Model Checking
A4
An FBD program can be checked by model checking tools
A5
A Verilog program can be checked by model checking tools
S6
Argument By satisfying the safety properties
G9
Claim There is a technique to enable model checking FBD programs
G10
Claim FBD programs satisfy the same safety properties with Verilog programs
G11
Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 2
Develop the 'FBDtoCadenceSMV' translator
Sn 3
Compare the model checking result of FBD programs with Verilog programs
Sn 4
Compare the model checking result of Verilog programs with FBD programs

2.1 The Safety Demonstration Strategy

The translator work

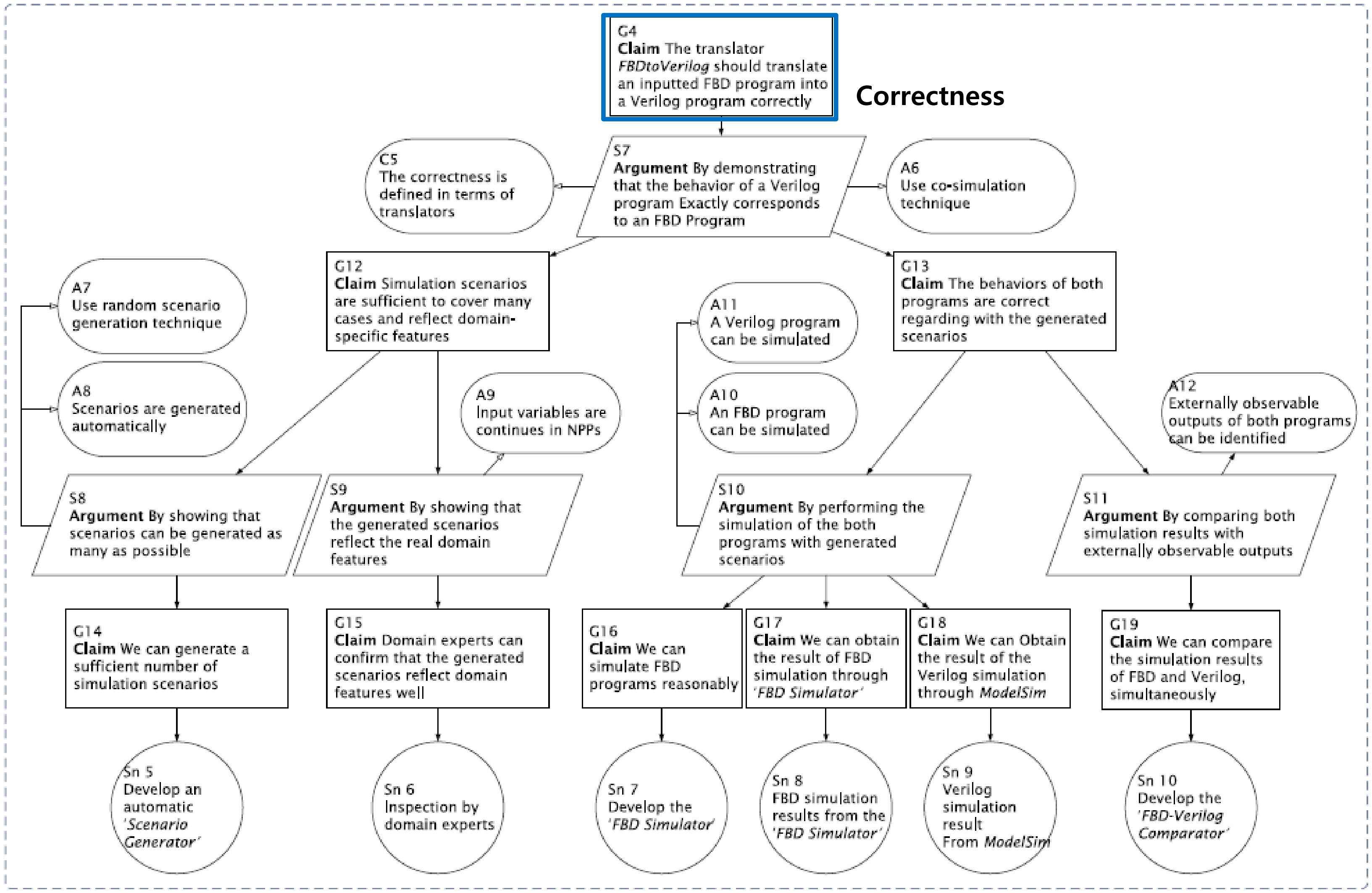

G3
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program safely
Safely
S3
Argument By demonstrating an FBD program and a Verilog program satisfy the same safety properties
C4
The safety is defined in terms of translators
A1
Use the model checking technique
G5
Claim The safety properties are appropriate
G6
Claim An inputted FBD program satisfies the safety properties
G7
Claim A translated Verilog program satisfies the same properties, too
S4
Argument By showing safety properties are exactly reflecting the safety of an inputted FBD program
S5
Argument By showing safety properties are written in temporal logic precisely
S6
Argument By satisfying the safety properties
A4
An FBD program can be checked by model checking tools
A5
A Verilog program can be checked by model checking tools
A2
Safety properties are developed through safety analysis or by domain experts
G8
Claim Confirm whether the properties reflect the safety of target system and are well defined with formalism or not
A3
Safety properties are written in temporal logic for the model checker to use
G9
Claim There is a technique to enable model checking FBD programs
G10
Claim FBD programs satisfy the same safety properties with Verilog programs
G11
Claim Verilog programs satisfy the same safety properties with FBD programs
Sn 1
Inspection by experts about each domain system and formalism
Safety Property
Sn 2
Develop the 'FBDtoCadenceSMV' translator
Sn 3
Compare the model checking result of FBD programs with Verilog programs
Sn 4
Compare the model checking result of Verilog programs with FBD programs
Model Checking

then

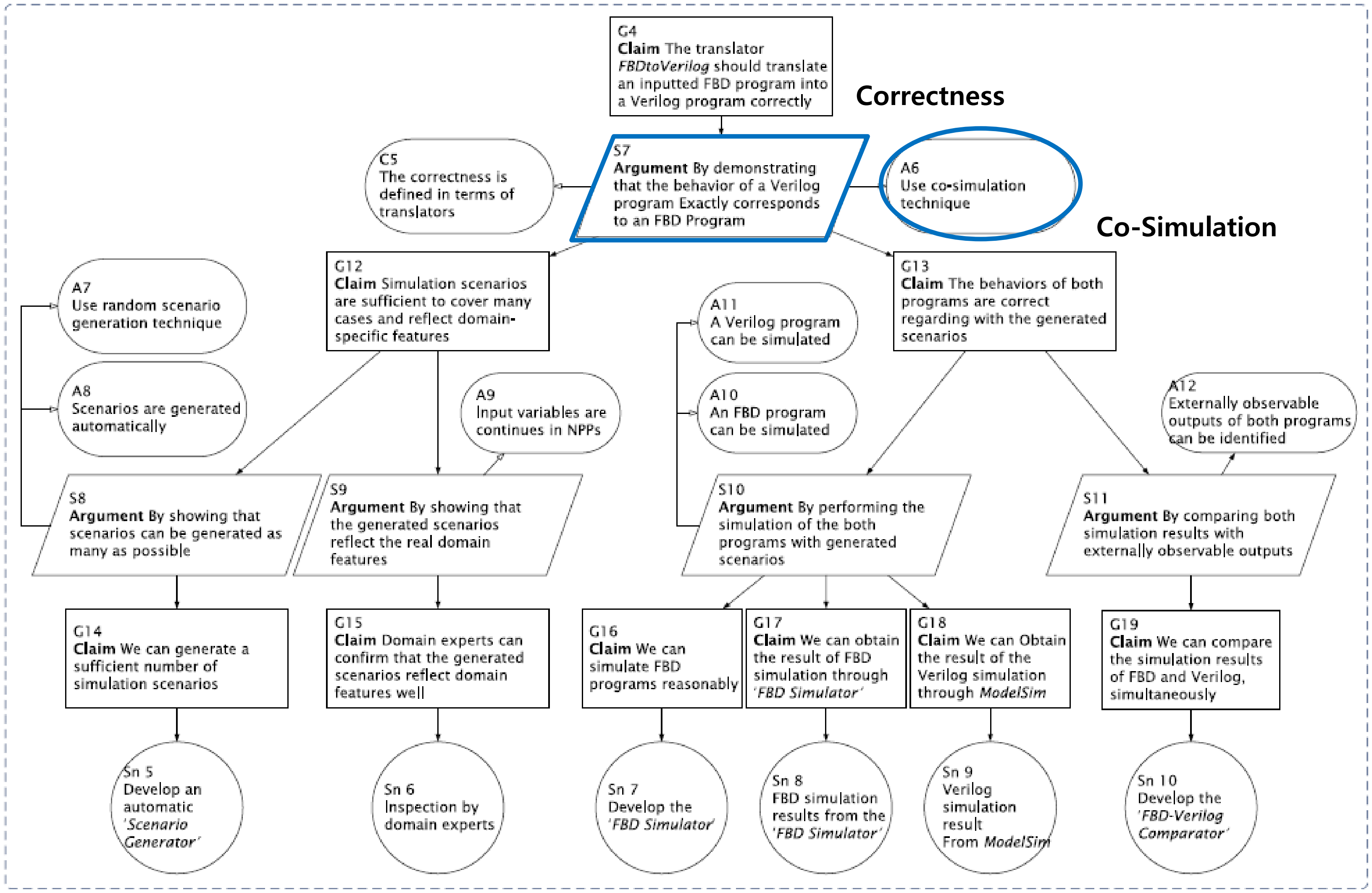
if

&

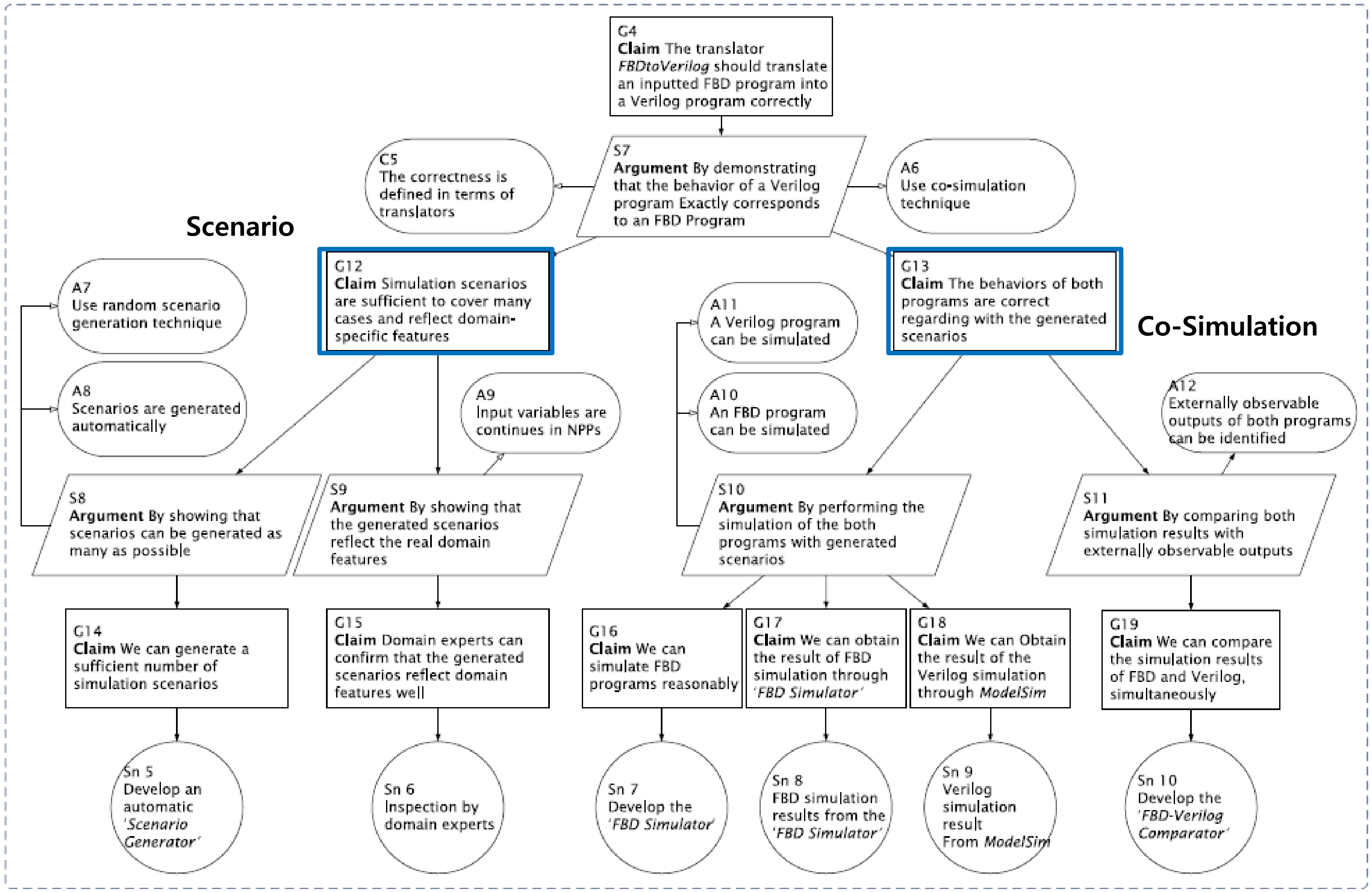
2.2 The Correctness Demonstration Strategy


G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program correctly
Correctness
C5
The correctness is defined in terms of translators
S7
Argument By demonstrating that the behavior of a Verilog program Exactly corresponds to an FBD Program
A6
Use co-simulation technique
G12
Claim Simulation scenarios are sufficient to cover many cases and reflect domain-specific features
G13
Claim The behaviors of both programs are correct regarding with the generated scenarios
A7
Use random scenario generation technique
A8
Scenarios are generated automatically
A9
Input variables are continues in NPPs
A11
A Verilog program can be simulated
A10
An FBD program can be simulated
A12
Externally observable outputs of both programs can be identified
S8
Argument By showing that scenarios can be generated as many as possible
S9
Argument By showing that the generated scenarios reflect the real domain features
S10
Argument By performing the simulation of the both programs with generated scenarios
S11
Argument By comparing both simulation results with externally observable outputs
G14
Claim We can generate a sufficient number of simulation scenarios
G15
Claim Domain experts can confirm that the generated scenarios reflect domain features well
G16
Claim We can simulate FBD programs reasonably
G17
Claim We can obtain the result of FBD simulation through 'FBD Simulator'
G18
Claim We can Obtain the result of the Verilog simulation through ModelSim
G19
Claim We can compare the simulation results of FBD and Verilog, simultaneously
Sn 5
Develop an automatic 'Scenario Generator'
Sn 6
Inspection by domain experts
Sn 7
Develop the 'FBD Simulator'
Sn 8
FBD simulation results from the 'FBD Simulator'
Sn 9
Verilog simulation result From ModelSim
Sn 10
Develop the 'FBD-Verilog Comparator'

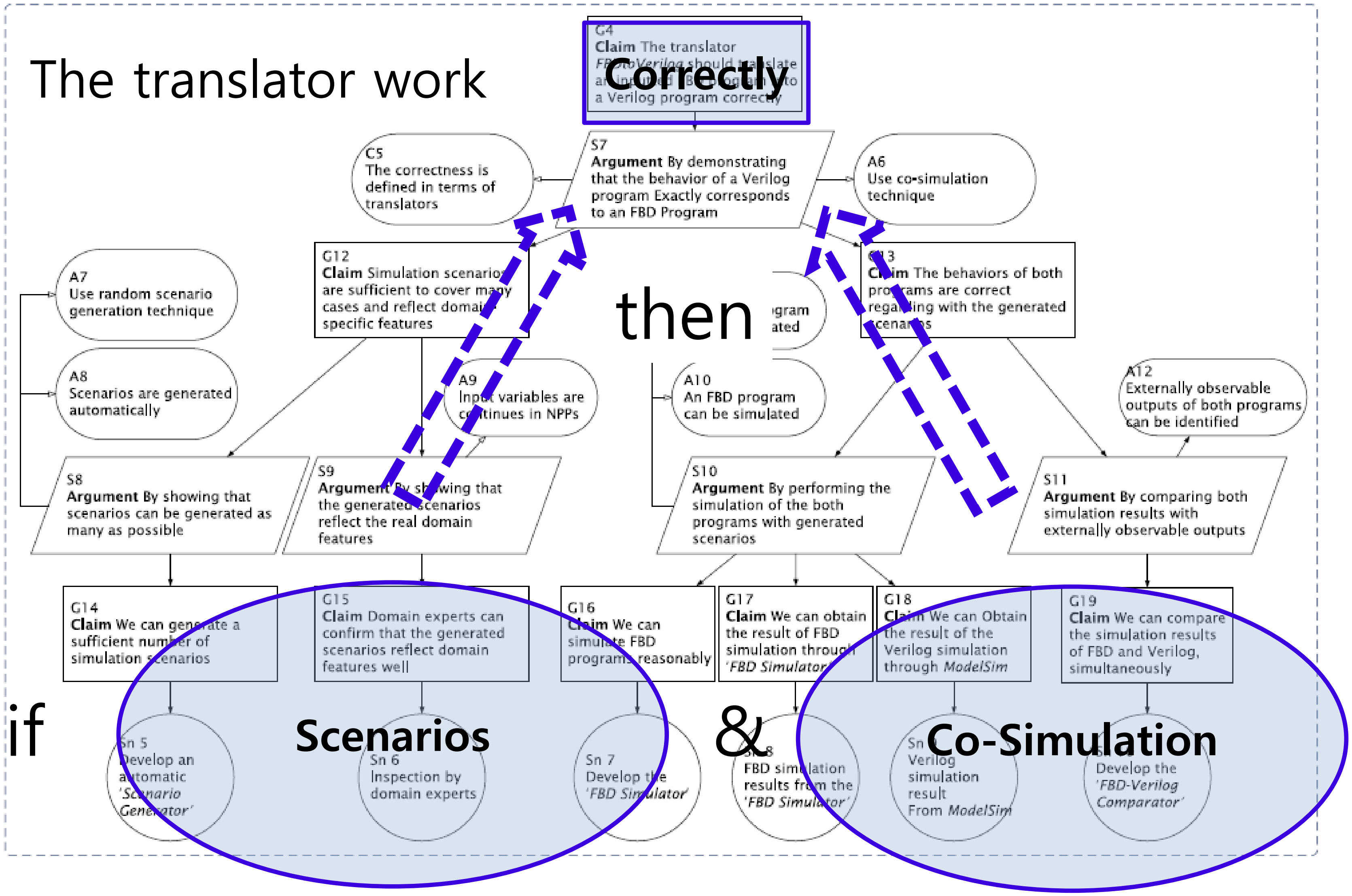
2.2 The Correctness Demonstration Strategy


G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program correctly
Correctness
C5
The correctness is defined in terms of translators
S7
Argument By demonstrating that the behavior of a Verilog program Exactly corresponds to an FBD Program
A6
Use co-simulation technique
Co-Simulation
A7
Use random scenario generation technique
A8
Scenarios are generated automatically
G12
Claim Simulation scenarios are sufficient to cover many cases and reflect domain-specific features
A9
Input variables are continues in NPPs
A11
A Verilog program can be simulated
A10
An FBD program can be simulated
G13
Claim The behaviors of both programs are correct regarding with the generated scenarios
A12
Externally observable outputs of both programs can be identified
S8
Argument By showing that scenarios can be generated as many as possible
S9
Argument By showing that the generated scenarios reflect the real domain features
S10
Argument By performing the simulation of the both programs with generated scenarios
S11
Argument By comparing both simulation results with externally observable outputs
G14
Claim We can generate a sufficient number of simulation scenarios
G15
Claim Domain experts can confirm that the generated scenarios reflect domain features well
G16
Claim We can simulate FBD programs reasonably
G17
Claim We can obtain the result of FBD simulation through 'FBD Simulator'
G18
Claim We can Obtain the result of the Verilog simulation through ModelSim
G19
Claim We can compare the simulation results of FBD and Verilog, simultaneously
Sn 5
Develop an automatic 'Scenario Generator'
Sn 6
Inspection by domain experts
Sn 7
Develop the 'FBD Simulator'
Sn 8
FBD simulation results from the 'FBD Simulator'
Sn 9
Verilog simulation result From ModelSim
Sn 10
Develop the 'FBD-Verilog Comparator'

2.2 The Correctness Demonstration Strategy

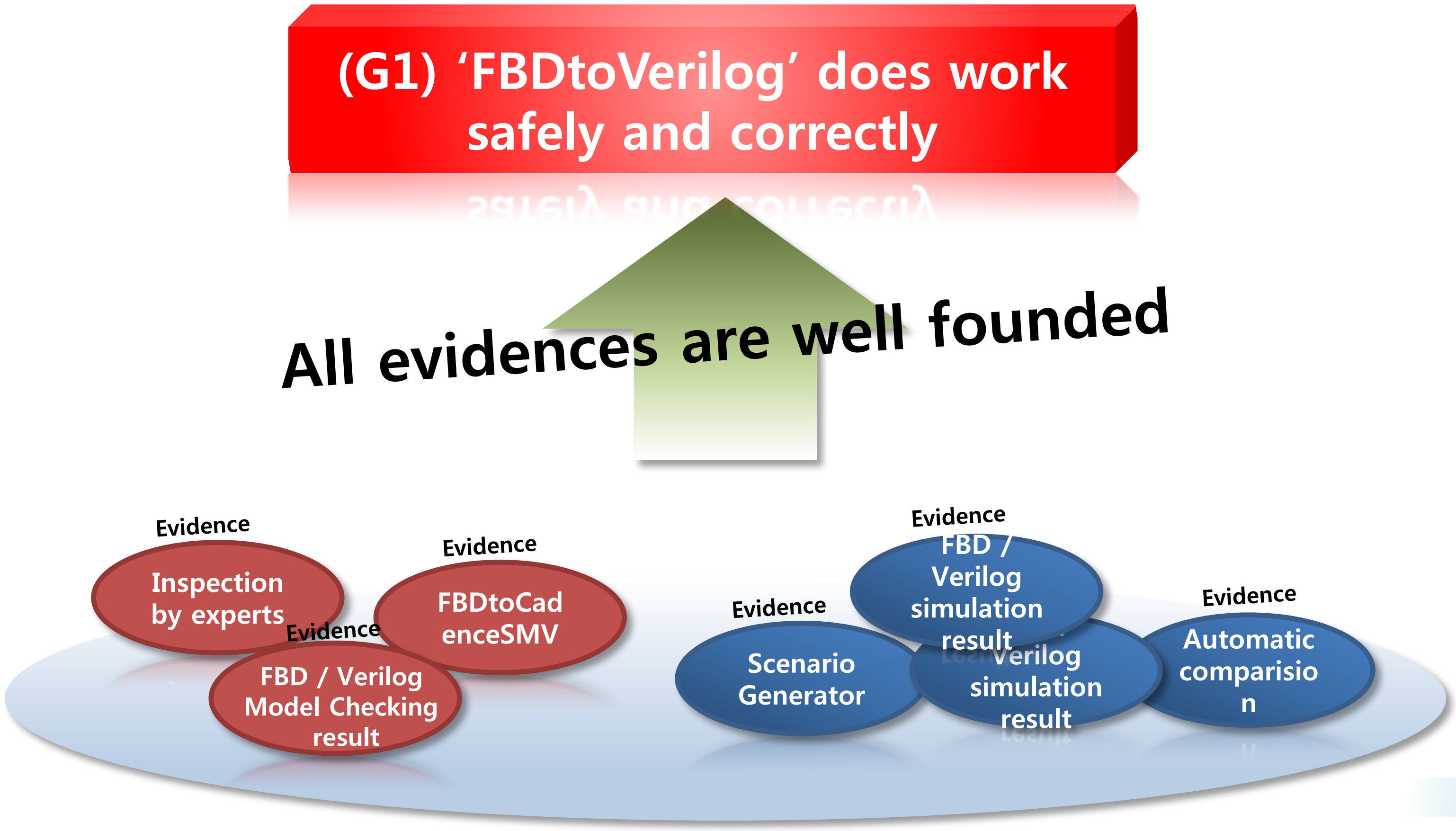

G4
Claim The translator FBDtoVerilog should translate an inputted FBD program into a Verilog program correctly
C5
The correctness is defined in terms of translators
S7
Argument By demonstrating that the behavior of a Verilog program Exactly corresponds to an FBD Program
A6
Use co-simulation technique
Scenario
Co-Simulation
G12
Claim Simulation scenarios are sufficient to cover many cases and reflect domain-specific features
G13
Claim The behaviors of both programs are correct regarding with the generated scenarios
A7
Use random scenario generation technique
A8
Scenarios are generated automatically
A9
Input variables are continues in NPPs
A11
A Verilog program can be simulated
A10
An FBD program can be simulated
A12
Externally observable outputs of both programs can be identified
S8
Argument By showing that scenarios can be generated as many as possible
S9
Argument By showing that the generated scenarios reflect the real domain features
S10
Argument By performing the simulation of the both programs with generated scenarios
S11
Argument By comparing both simulation results with externally observable outputs
G14
Claim We can generate a sufficient number of simulation scenarios
G15
Claim Domain experts can confirm that the generated scenarios reflect domain features well
G16
Claim We can simulate FBD programs reasonably
G17
Claim We can obtain the result of FBD simulation through 'FBD Simulator'
G18
Claim We can Obtain the result of the Verilog simulation through ModelSim
G19
Claim We can compare the simulation results of FBD and Verilog, simultaneously
Sn 5
Develop an automatic 'Scenario Generator'
Sn 6
Inspection by domain experts
Sn 7
Develop the 'FBD Simulator'
Sn 8
FBD simulation results from the 'FBD Simulator'
Sn 9
Verilog simulation result From ModelSim
Sn 10
Develop the 'FBD-Verilog Comparator'

2.2 The Correctness Demonstration Strategy


The translator work
Correctly
then
if
Scenarios
&
Co-Simulation
G4 Claim The translator FBDtoVerilog should translate an FBD program to a Verilog program correctly
S7 Argument By demonstrating that the behavior of a Verilog program Exactly corresponds to an FBD Program
C5 The correctness is defined in terms of translators
A6 Use co-simulation technique
G12 Claim Simulation scenarios are sufficient to cover many cases and reflect domain specific features
G13 Claim The behaviors of both programs are correct regarding with the generated scenarios
A7 Use random scenario generation technique
A8 Scenarios are generated automatically
A9 Input variables are continues in NPPs
A10 An FBD program can be simulated
A12 Externally observable outputs of both programs can be identified
S8 Argument By showing that scenarios can be generated as many as possible
S9 Argument By showing that the generated scenarios reflect the real domain features
S10 Argument By performing the simulation of the both programs with generated scenarios
S11 Argument By comparing both simulation results with externally observable outputs
G14 Claim We can generate a sufficient number of simulation scenarios
G15 Claim Domain experts can confirm that the generated scenarios reflect domain features well
G16 Claim We can simulate FBD programs reasonably
G17 Claim We can obtain the result of FBD simulation through 'FBD Simulator'
G18 Claim We can Obtain the result of the Verilog simulation through ModelSim
G19 Claim We can compare the simulation results of FBD and Verilog, simultaneously
Sn 5 Develop an automatic 'Scenario Generator'
Sn 6 Inspection by domain experts
Sn 7 Develop the 'FBD Simulator'
Sn 8 FBD simulation results from the 'FBD Simulator'
Sn Verilog simulation result From ModelSim
Sn Develop the 'FBD-Verilog Comparator'

In summary

- In summary, we constructed our purpose and strategy with the GSN.
  - We first set up the top-level goal (G1) and divided it into two parts, **safety (G3)** and **correctness (G4)**, then presented sub-goals, arguments and evidences to accomplish upper goals.

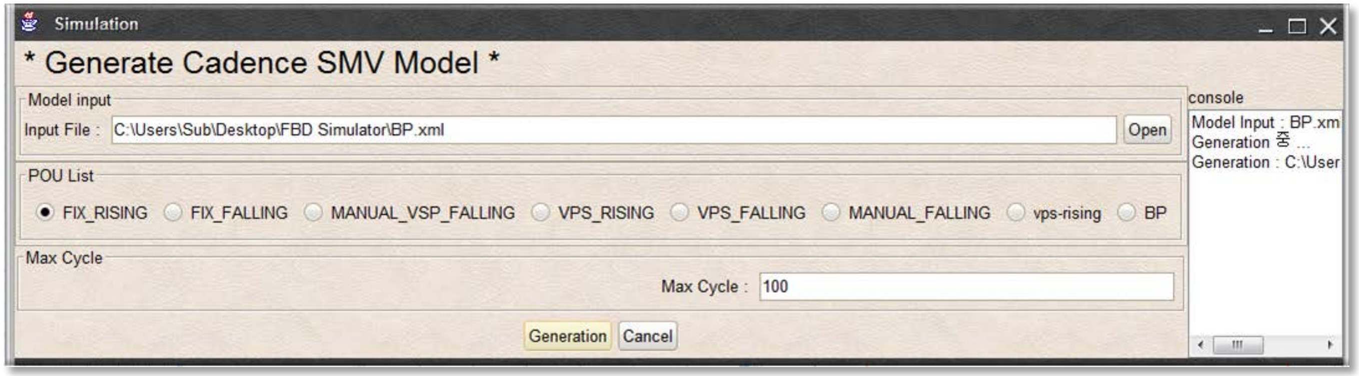

(G1) 'FBDtoVerilog' does work safely and correctly
All evidences are well founded
Evidence
Inspection by experts
Evidence
FBDtoCadenceSMV
Evidence
FBD / Verilog Model Checking result
Evidence
Scenario Generator
Evidence
FBD / Verilog simulation result
Verilog simulation result
Evidence
Automatic comparision

1. FBDtoCadenceSMV
2. Scenario Generator
3. FBD Simulator
4. FBD-Verilog Comparator

3. THE DEVELOPMENT OF SUPPORTING TOOLS

3.1 FBDtoCadenceSMV

- FBD program → input program of Cadence SMV (Model checker)
  (Translate)



Simulation

* Generate Cadence SMV Model *

Model input

Input File : C:\Users\Sub\Desktop\FBD Simulator\BP.xml [Open]

POU List

[✓ FIX_RISING] FIX_FALLING MANUAL_VSP_FALLING VPS_RISING VPS_FALLING MANUAL_FALLING vps-rising BP

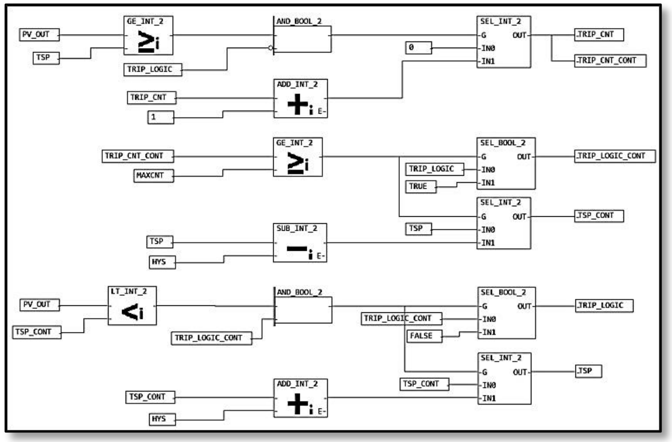
Max Cycle

Max Cycle : 100

[Generation] [Cancel]

console

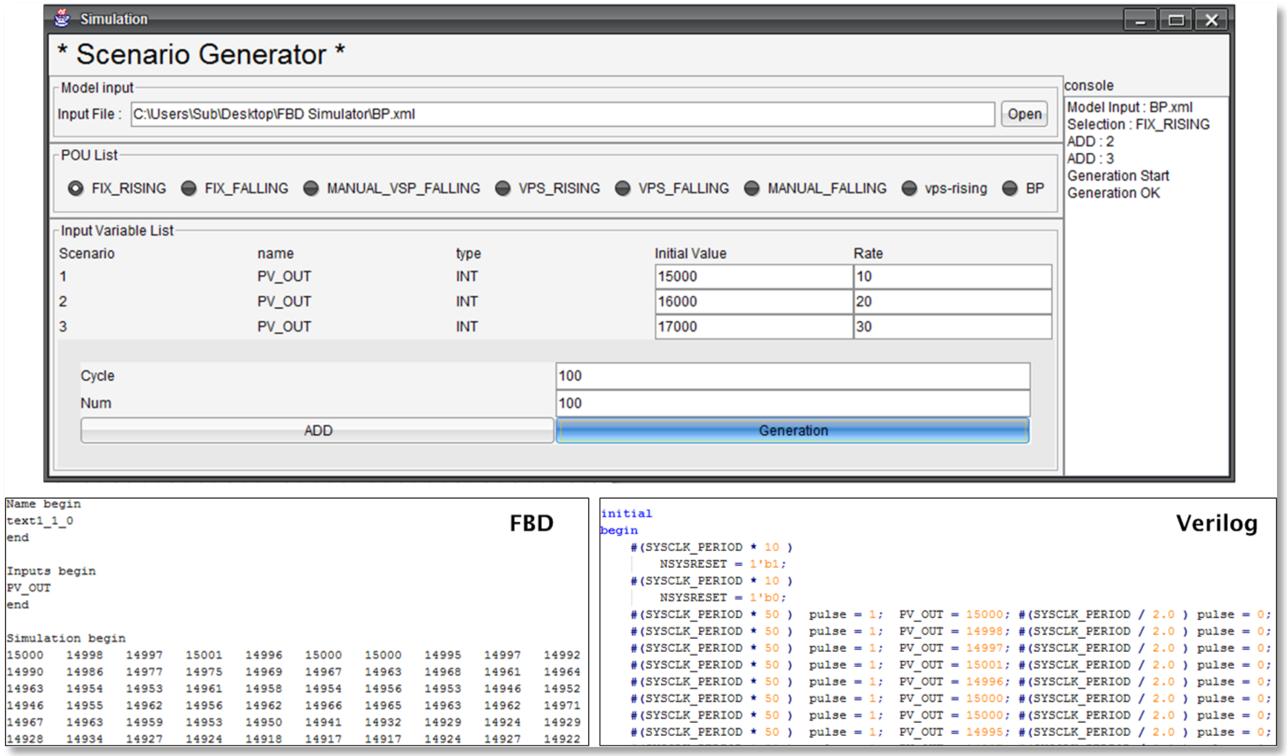
```
Model Input : BP.xm
Generation 중 ...
Generation : C:\User
```



```
module example (in, clk, reset, out);
input in, clk, reset;
output out;

wire in, clk, reset;
reg mid1, mid2, out;

always @ (posedge clk or negedge reset) begin
        if(!reset) begin
                mid1 <= 0;
                mid2 <= 0;
                out <= 0;
        end
        else begin
                mid1 <= in;
                mid2 <= mid1;

                if(in==1 && mid1==1 && mid2==1) begin
                        out <= 1;
                end
                else begin
                        out <= 0;
                end
        end
end
endmodule
```

3.2 Scenario Generator

- The *'Scenario Generator'* **randomly generates** a number of scenarios **within predefined constraints on input values.**



Model input — Input File : C:\Users\Sub\Desktop\FBD Simulator\BP.xml

POU List: [✓ FIX_RISING] FIX_FALLING MANUAL_VSP_FALLING VPS_RISING VPS_FALLING MANUAL_FALLING vps-rising BP

Input Variable List

Scenario	name	type	Initial Value	Rate
1	PV_OUT	INT	15000	10
2	PV_OUT	INT	16000	20
3	PV_OUT	INT	17000	30

Cycle	100
Num	100

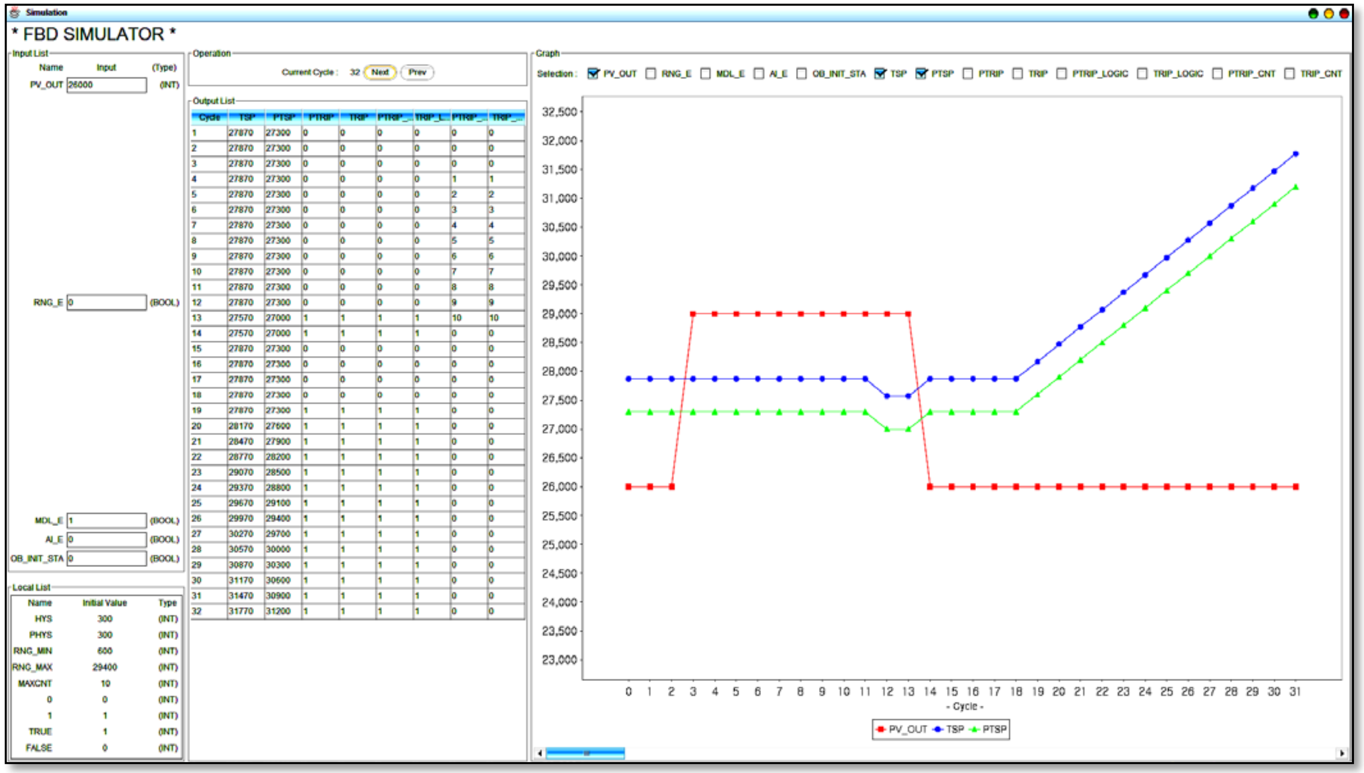
```
console
Model Input : BP.xml
Selection : FIX_RISING
ADD : 2
ADD : 3
Generation Start
Generation OK
```

```
Name begin
text1_1_0
end

Inputs begin
PV_OUT
end

Simulation begin
15000   14998   14997   15001   14996   15000   15000   14995   14997   14992
14990   14986   14977   14975   14969   14967   14963   14968   14961   14964
14963   14954   14953   14961   14958   14954   14956   14953   14946   14952
14946   14955   14962   14956   14962   14966   14965   14963   14962   14971
14967   14963   14959   14953   14950   14941   14932   14929   14924   14929
14928   14934   14927   14924   14918   14917   14917   14924   14927   14922
```

```
initial
begin
    #(SYSCLK_PERIOD * 10 )
        NSYSRESET = 1'b1;
    #(SYSCLK_PERIOD * 10 )
        NSYSRESET = 1'b0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 15000; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 14998; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 14997; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 15001; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 14996; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 15000; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 15000; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
    #(SYSCLK_PERIOD * 50 )  pulse = 1;  PV_OUT = 14995; #(SYSCLK_PERIOD / 2.0 ) pulse = 0;
```

3.3 FBD Simulator

- The *'FBD Simulator'* works in two modes.
  - This FBD Simulator executes one scenario and visualizes the results in a form of graphical chart.
  - It support a verification of functionality of FBD.



Simulation
* FBD SIMULATOR *
Operation
Current Cycle : 32
Next
Prev
Graph
Selection : [✓ PV_OUT] RNG_E MDL_E AI_E OB_INIT_STA [✓ TSP] [✓ PTSP] PTRIP TRIP PTRIP_LOGIC TRIP_LOGIC PTRIP_CNT TRIP_CNT
- Cycle -
PV_OUT
TSP
PTSP

Input List

Name	Input	(Type)
PV_OUT	26000	(INT)
RNG_E	0	(BOOL)
MDL_E	1	(BOOL)
AI_E	0	(BOOL)
OB_INIT_STA	0	(BOOL)

Output List

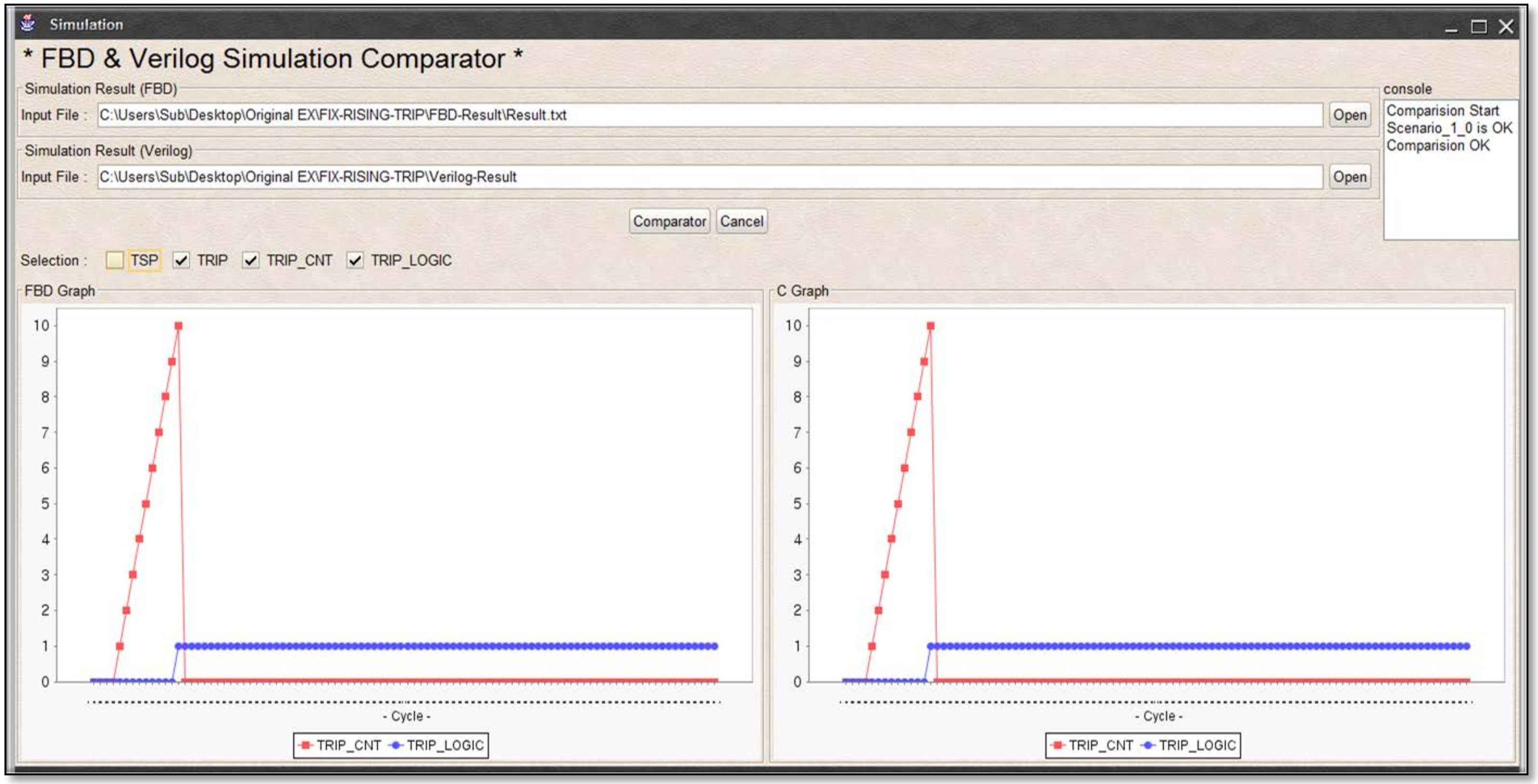
Cycle	TSP	PTSP	PTRIP	TRIP	PTRIP_...	TRIP_L...	PTRIP_...	TRIP_...
1	27870	27300	0	0	0	0	0	0
2	27870	27300	0	0	0	0	0	0
3	27870	27300	0	0	0	0	0	0
4	27870	27300	0	0	0	0	1	1
5	27870	27300	0	0	0	0	2	2
6	27870	27300	0	0	0	0	3	3
7	27870	27300	0	0	0	0	4	4
8	27870	27300	0	0	0	0	5	5
9	27870	27300	0	0	0	0	6	6
10	27870	27300	0	0	0	0	7	7
11	27870	27300	0	0	0	0	8	8
12	27870	27300	0	0	0	0	9	9
13	27570	27000	1	1	1	1	10	10
14	27570	27000	1	1	1	1	0	0
15	27870	27300	0	0	0	0	0	0
16	27870	27300	0	0	0	0	0	0
17	27870	27300	0	0	0	0	0	0
18	27870	27300	0	0	0	0	0	0
19	27870	27300	1	1	1	1	0	0
20	28170	27600	1	1	1	1	0	0
21	28470	27900	1	1	1	1	0	0
22	28770	28200	1	1	1	1	0	0
23	29070	28500	1	1	1	1	0	0
24	29370	28800	1	1	1	1	0	0
25	29670	29100	1	1	1	1	0	0
26	29970	29400	1	1	1	1	0	0
27	30270	29700	1	1	1	1	0	0
28	30570	30000	1	1	1	1	0	0
29	30870	30300	1	1	1	1	0	0
30	31170	30600	1	1	1	1	0	0
31	31470	30900	1	1	1	1	0	0
32	31770	31200	1	1	1	1	0	0

Local List

Name	Initial Value	Type
HYS	300	(INT)
PHYS	300	(INT)
RNG_MIN	600	(INT)
RNG_MAX	29400	(INT)
MAXCNT	10	(INT)
0	0	(INT)
1	1	(INT)
TRUE	1	(INT)
FALSE	0	(INT)

3.4 FBD-Verilog Comparator

- Automatic comparison between FBD simulation and Verilog simulation results.


Simulation

* FBD & Verilog Simulation Comparator *

Simulation Result (FBD)

Input File : C:\Users\Sub\Desktop\Original EX\FIX-RISING-TRIP\FBD-Result\Result.txt

Simulation Result (Verilog)

Input File : C:\Users\Sub\Desktop\Original EX\FIX-RISING-TRIP\Verilog-Result

console

```
Comparision Start
Scenario_1_0 is OK
Comparision OK
```

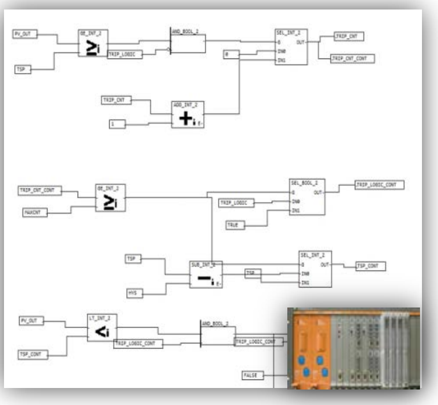
Selection : ☐ TSP ✓ TRIP ✓ TRIP_CNT ✓ TRIP_LOGIC

1. The Safety Demonstration
2. The Correctness Demonstration

4. CASE STUDY

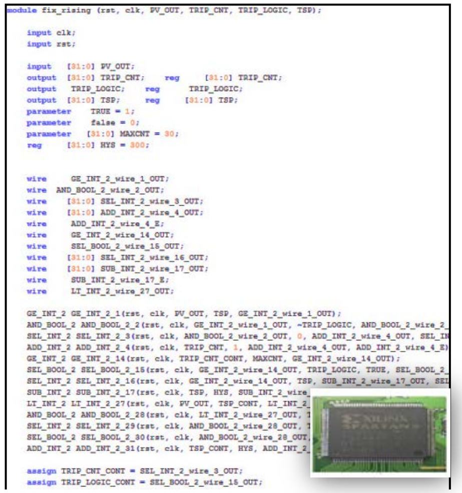
4. Case Study

- KNIC project RPS (Reactor Protection System) BP (Bistable Process)



FBD program for **PLC**

FBDtoVerilog

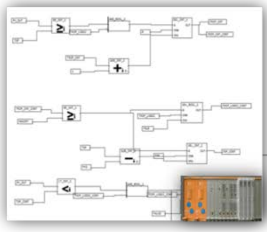


```
module fix_rising (rst, clk, PV_OUT, TRIP_CNT, TRIP_LOGIC, TSP);

   input clk;
   input rst;

   input   [31:0] PV_OUT;
   output  [31:0] TRIP_CNT;    reg     [31:0] TRIP_CNT;
   output   TRIP_LOGIC;    reg      TRIP_LOGIC;
   output  [31:0] TSP;     reg     [31:0] TSP;
   parameter    TRUE = 1;
   parameter    false = 0;
   parameter   [31:0] MAXCNT = 30;
   reg     [31:0] HYS = 300;


   wire     GE_INT_2_wire_1_OUT;
   wire  AND_BOOL_2_wire_2_OUT;
   wire    [31:0] SEL_INT_2_wire_3_OUT;
   wire    [31:0] ADD_INT_2_wire_4_OUT;
   wire     ADD_INT_2_wire_4_E;
   wire     GE_INT_2_wire_14_OUT;
   wire     SEL_BOOL_2_wire_15_OUT;
   wire    [31:0] SEL_INT_2_wire_16_OUT;
   wire    [31:0] SUB_INT_2_wire_17_OUT;
   wire     SUB_INT_2_wire_17_E;
   wire     LT_INT_2_wire_27_OUT;

   GE_INT_2 GE_INT_2_1(rst, clk, PV_OUT, TSP, GE_INT_2_wire_1_OUT);
   AND_BOOL_2 AND_BOOL_2_2(rst, clk, GE_INT_2_wire_1_OUT, ~TRIP_LOGIC, AND_BOOL_2_wire_2_
   SEL_INT_2 SEL_INT_2_3(rst, clk, AND_BOOL_2_wire_2_OUT, 0, ADD_INT_2_wire_4_OUT, SEL_IN
   ADD_INT_2 ADD_INT_2_4(rst, clk, TRIP_CNT, 1, ADD_INT_2_wire_4_OUT, ADD_INT_2_wire_4_E)
   GE_INT_2 GE_INT_2_14(rst, clk, TRIP_CNT_CONT, MAXCNT, GE_INT_2_wire_14_OUT);
   SEL_BOOL_2 SEL_BOOL_2_15(rst, clk, GE_INT_2_wire_14_OUT, TRIP_LOGIC, TRUE, SEL_BOOL_2_
   SEL_INT_2 SEL_INT_2_16(rst, clk, GE_INT_2_wire_14_OUT, TSP, SUB_INT_2_wire_17_OUT, SEL
   SUB_INT_2 SUB_INT_2_17(rst, clk, TSP, HYS, SUB_INT_2_wire_
   LT_INT_2 LT_INT_2_27(rst, clk, PV_OUT, TSP_CONT, LT_INT_2_
   AND_BOOL_2 AND_BOOL_2_28(rst, clk, LT_INT_2_wire_27_OUT, T
   SEL_INT_2 SEL_INT_2_29(rst, clk, AND_BOOL_2_wire_28_OUT, T
   SEL_BOOL_2 SEL_BOOL_2_30(rst, clk, AND_BOOL_2_wire_28_OUT,
   ADD_INT_2 ADD_INT_2_31(rst, clk, TSP_CONT, HYS, ADD_INT_2_

   assign TRIP_CNT_CONT = SEL_INT_2_wire_3_OUT;
   assign TRIP_LOGIC_CONT = SEL_BOOL_2_wire_15_OUT;
```

Verilog program for **FPGA**

4.1 The Safety Demonstration (G3)

- We performed **model checking** with the **Cadence SMV**
- We developed 28 safety properties with assistant from domain exports and referable papers.
- Ex)
  - *"If PV_OUT (An input sensor value) is more than the TSP (Trip Set-Point) for a predefined time, then the trip signal should be fired (TRIP_LOGIC = 1) immediately.*"
  - : AG((PV_OUT > TSP) & (TRIP_CNT >= (MAXCNT - 1)) → AX(TRIP_LOGIC = 1))

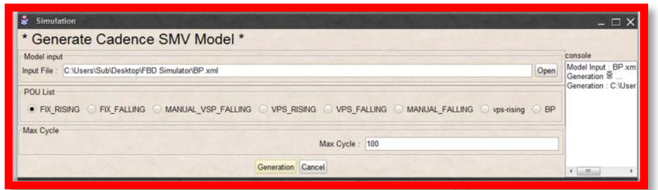


FBD program for **PLC**

Correct?

FBDtoVerilog

Verilog program for **FPGA**



* Generate Cadence SMV Model *

Model input

Input File : C:\Users\Sub\Desktop\FBD Simulator\BP.xml

POU List

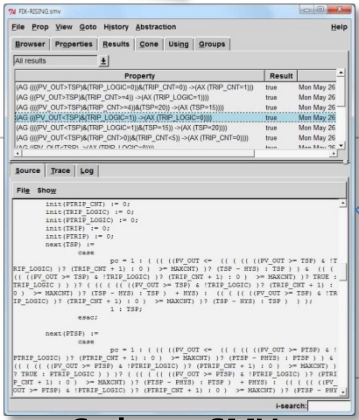
[✓ FIX_RISING] FIX_FALLING MANUAL_VSP_FALLING VPS_RISING VPS_FALLING MANUAL_FALLING vps-rising BP

Max Cycle

Max Cycle : 100

```
Model Input : BP.xm
Generation 중 ...
Generation : C:\User
```

Safety Properties


FIX-RISING.smv
File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Source Trace Log
File Show
i-search:

Property	Result	
(AG ((((PV_OUT>TSP)&(TRIP_LOGIC=0))&(TRIP_CNT=0)) ->(AX (TRIP_CNT=1))))	true	Mon May 26
(AG (((PV_OUT>TSP)&(TRIP_CNT>=4)) ->(AX (TRIP_LOGIC=1))))	true	Mon May 26
(AG ((((PV_OUT>TSP)&(TRIP_CNT>=4))&(TSP=20)) ->(AX (TSP=15))))	true	Mon May 26
(AG (((PV_OUT<TSP)&(TRIP_LOGIC=1)) ->(AX (TRIP_LOGIC=0))))	true	Mon May 26
(AG ((((PV_OUT<TSP)&(TRIP_LOGIC=1))&(TSP=15)) ->(AX (TSP=20))))	true	Mon May 26
(AG ((((PV_OUT<TSP)&(TRIP_CNT>0))&(TRIP_CNT<5)) ->(AX (TRIP_CNT=0))))	true	Mon May 26

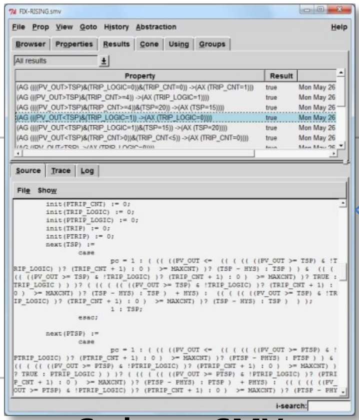
```
        init(PTRIP_CNT) := 0;
        init(TRIP_LOGIC) := 0;
        init(PTRIP_LOGIC) := 0;
        init(TRIP) := 0;
        init(PTRIP) := 0;
        next(TSP) :=
                case
                        pc = 1 : ( (( ((PV_OUT <=  (( ( (( ((PV_OUT >= TSP) & !T
RIP_LOGIC) )? (TRIP_CNT + 1) : 0 )  >= MAXCNT) )? (TSP - HYS) : TSP ) ) &  (( (
(( ((PV_OUT >= TSP) & !TRIP_LOGIC) )? (TRIP_CNT + 1) : 0 )  >= MAXCNT) )? TRUE :
TRIP_LOGIC ) ) )? ( (( ( (( ((PV_OUT >= TSP) & !TRIP_LOGIC) )? (TRIP_CNT + 1) :
0 )  >= MAXCNT) )? (TSP - HYS) : TSP )  + HYS) :  (( ( (( ((PV_OUT >= TSP) & !TR
IP_LOGIC) )? (TRIP_CNT + 1) : 0 )  >= MAXCNT) )? (TSP - HYS) : TSP )  ) );
                        1 : TSP;
                esac;

        next(PTSP) :=
                case
                        pc = 1 : ( (( ((PV_OUT <=  (( ( (( ((PV_OUT >= PTSP) & !
PTRIP_LOGIC) )? (PTRIP_CNT + 1) : 0 )  >= MAXCNT) )? (PTSP - PHYS) : PTSP ) ) &
(( ( (( ((PV_OUT >= PTSP) & !PTRIP_LOGIC) )? (PTRIP_CNT + 1) : 0 )  >= MAXCNT) )
? TRUE : PTRIP_LOGIC ) ) )? ( (( ( (( ((PV_OUT >= PTSP) & !PTRIP_LOGIC) )? (PTRI
P_CNT + 1) : 0 )  >= MAXCNT) )? (PTSP - PHYS) : PTSP )  + PHYS) :  (( ( (( ((PV_
OUT >= PTSP) & !PTRIP_LOGIC) )? (PTRIP_CNT + 1) : 0 )  >= MAXCNT) )? (PTSP - PHY
```

Cadence SMV

<u>True</u>

False

Counter Example


FIX-RISING.smv
File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Source Trace Log
File Show
i-search:

Property	Result	
(AG ((((PV_OUT>TSP)&(TRIP_LOGIC=0))&(TRIP_CNT=0)) ->(AX (TRIP_CNT=1))))	true	Mon May 26
(AG (((PV_OUT>TSP)&(TRIP_CNT>=4)) ->(AX (TRIP_LOGIC=1))))	true	Mon May 26
(AG ((((PV_OUT>TSP)&(TRIP_CNT>=4))&(TSP=20)) ->(AX (TSP=15))))	true	Mon May 26
(AG (((PV_OUT<TSP)&(TRIP_LOGIC=1)) ->(AX (TRIP_LOGIC=0))))	true	Mon May 26
(AG ((((PV_OUT<TSP)&(TRIP_LOGIC=1))&(TSP=15)) ->(AX (TSP=20))))	true	Mon May 26
(AG ((((PV_OUT<TSP)&(TRIP_CNT>0))&(TRIP_CNT<5)) ->(AX (TRIP_CNT=0))))	true	Mon May 26

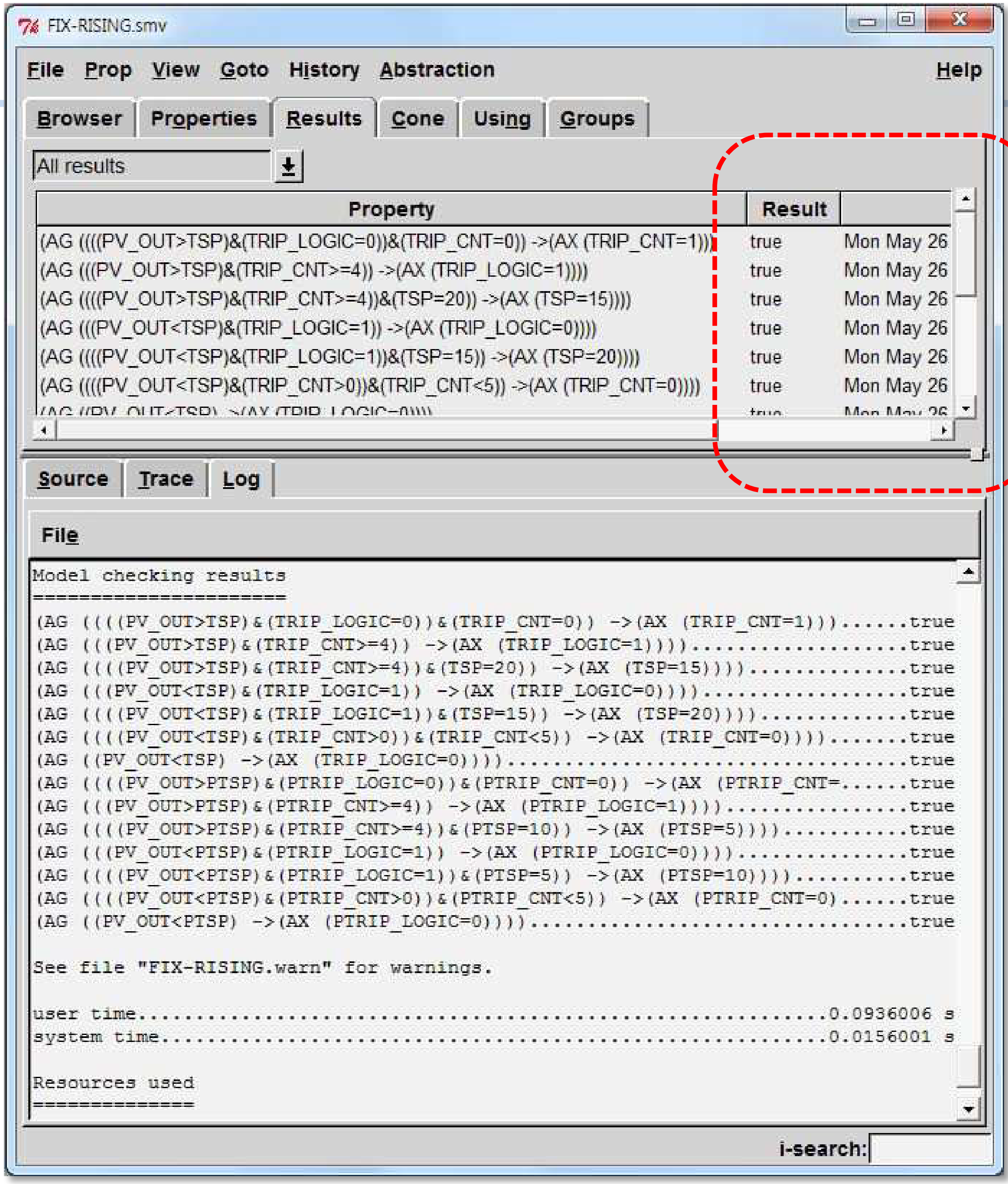
```
        init(PTRIP_CNT) := 0;
        init(TRIP_LOGIC) := 0;
        init(PTRIP_LOGIC) := 0;
        init(TRIP) := 0;
        init(PTRIP) := 0;
        next(TSP) :=
                case
                        pc = 1 : ( (( ((PV_OUT <=  (( ( (( ((PV_OUT >= TSP) & !T
RIP_LOGIC) )? (TRIP_CNT + 1) : 0 )  >= MAXCNT) )? (TSP - HYS) : TSP ) ) &  (( (
(( ((PV_OUT >= TSP) & !TRIP_LOGIC) )? (TRIP_CNT + 1) : 0 )  >= MAXCNT) )? TRUE :
TRIP_LOGIC ) ) )? ( (( ( (( ((PV_OUT >= TSP) & !TRIP_LOGIC) )? (TRIP_CNT + 1) :
0 )  >= MAXCNT) )? (TSP - HYS) : TSP )  + HYS) :  (( ( (( ((PV_OUT >= TSP) & !TR
IP_LOGIC) )? (TRIP_CNT + 1) : 0 )  >= MAXCNT) )? (TSP - HYS) : TSP )  ) );
                        1 : TSP;
                esac;

        next(PTSP) :=
                case
                        pc = 1 : ( (( ((PV_OUT <=  (( ( (( ((PV_OUT >= PTSP) & !
PTRIP_LOGIC) )? (PTRIP_CNT + 1) : 0 )  >= MAXCNT) )? (PTSP - PHYS) : PTSP ) ) &
(( ( (( ((PV_OUT >= PTSP) & !PTRIP_LOGIC) )? (PTRIP_CNT + 1) : 0 )  >= MAXCNT) )
? TRUE : PTRIP_LOGIC ) ) )? ( (( ( (( ((PV_OUT >= PTSP) & !PTRIP_LOGIC) )? (PTRI
P_CNT + 1) : 0 )  >= MAXCNT) )? (PTSP - PHYS) : PTSP )  + PHYS) :  (( ( (( ((PV_
OUT >= PTSP) & !PTRIP_LOGIC) )? (PTRIP_CNT + 1) : 0 )  >= MAXCNT) )? (PTSP - PHY
```

Cadence SMV

<u>True</u>

False

Counter Example

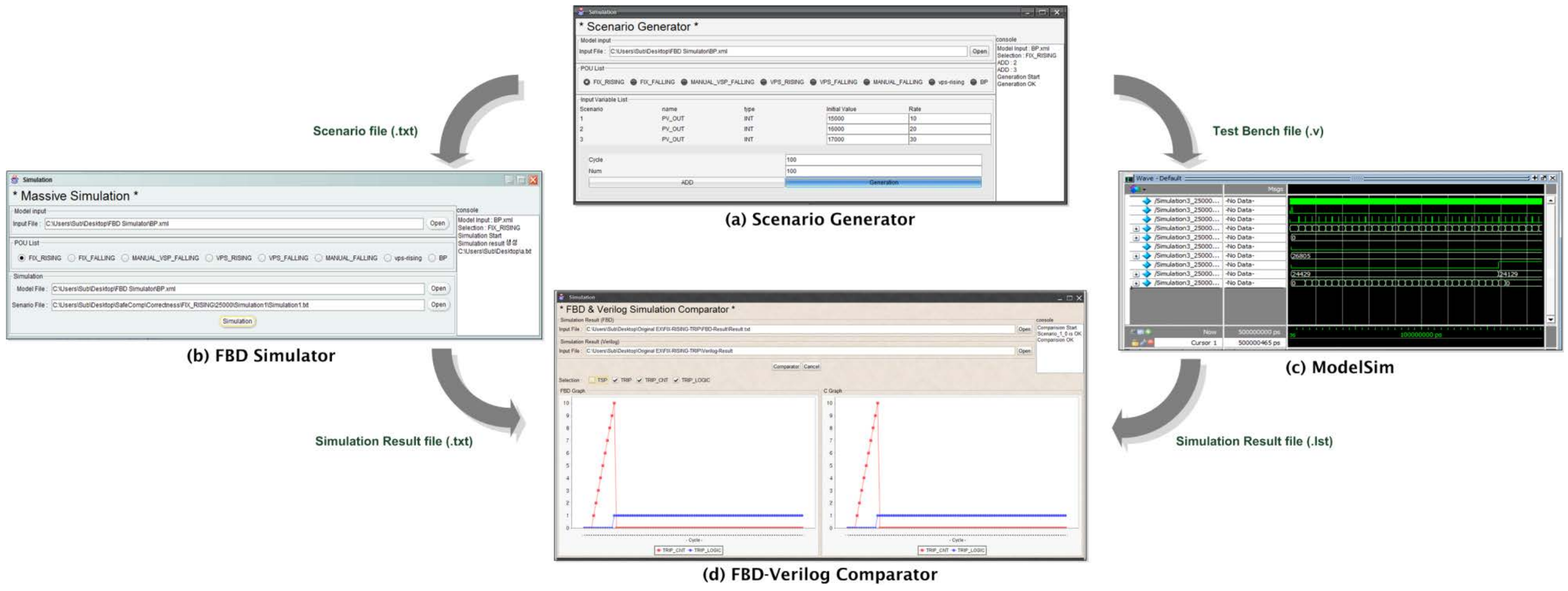

FIX-RISING.smv
File Prop View Goto History Abstraction Help
Browser Properties Results Cone Using Groups
All results
Source Trace Log
File
i-search:

Property	Result	
(AG ((((PV_OUT>TSP)&(TRIP_LOGIC=0))&(TRIP_CNT=0)) ->(AX (TRIP_CNT=1)))	true	Mon May 26
(AG (((PV_OUT>TSP)&(TRIP_CNT>=4)) ->(AX (TRIP_LOGIC=1))))	true	Mon May 26
(AG ((((PV_OUT>TSP)&(TRIP_CNT>=4))&(TSP=20)) ->(AX (TSP=15))))	true	Mon May 26
(AG (((PV_OUT<TSP)&(TRIP_LOGIC=1)) ->(AX (TRIP_LOGIC=0))))	true	Mon May 26
(AG ((((PV_OUT<TSP)&(TRIP_LOGIC=1))&(TSP=15)) ->(AX (TSP=20))))	true	Mon May 26
(AG ((((PV_OUT<TSP)&(TRIP_CNT>0))&(TRIP_CNT<5)) ->(AX (TRIP_CNT=0))))	true	Mon May 26
(AG ((PV_OUT<TSP) ->(AX (TRIP_LOGIC=0))))	true	Mon May 26

```
Model checking results
======================
(AG ((((PV_OUT>TSP)&(TRIP_LOGIC=0))&(TRIP_CNT=0)) ->(AX (TRIP_CNT=1)))......true
(AG (((PV_OUT>TSP)&(TRIP_CNT>=4)) ->(AX (TRIP_LOGIC=1))))....................true
(AG ((((PV_OUT>TSP)&(TRIP_CNT>=4))&(TSP=20)) ->(AX (TSP=15))))...............true
(AG (((PV_OUT<TSP)&(TRIP_LOGIC=1)) ->(AX (TRIP_LOGIC=0))))...................true
(AG ((((PV_OUT<TSP)&(TRIP_LOGIC=1))&(TSP=15)) ->(AX (TSP=20))))..............true
(AG ((((PV_OUT<TSP)&(TRIP_CNT>0))&(TRIP_CNT<5)) ->(AX (TRIP_CNT=0))))........true
(AG ((PV_OUT<TSP) ->(AX (TRIP_LOGIC=0))))....................................true
(AG ((((PV_OUT>PTSP)&(PTRIP_LOGIC=0))&(PTRIP_CNT=0)) ->(AX (PTRIP_CNT=......true
(AG (((PV_OUT>PTSP)&(PTRIP_CNT>=4)) ->(AX (PTRIP_LOGIC=1))))..................true
(AG ((((PV_OUT>PTSP)&(PTRIP_CNT>=4))&(PTSP=10)) ->(AX (PTSP=5))))............true
(AG (((PV_OUT<PTSP)&(PTRIP_LOGIC=1)) ->(AX (PTRIP_LOGIC=0))))................true
(AG ((((PV_OUT<PTSP)&(PTRIP_LOGIC=1))&(PTSP=5)) ->(AX (PTSP=10))))..........true
(AG ((((PV_OUT<PTSP)&(PTRIP_CNT>0))&(PTRIP_CNT<5)) ->(AX (PTRIP_CNT=0).......true
(AG ((PV_OUT<PTSP) ->(AX (PTRIP_LOGIC=0))))..................................true

See file "FIX-RISING.warn" for warnings.

user time..........................................................0.0936006 s
system time........................................................0.0156001 s

Resources used
==============
```

4.2 The Correctness Demonstration (G4)

- We performed **co-simulation** with co-simulation environment.



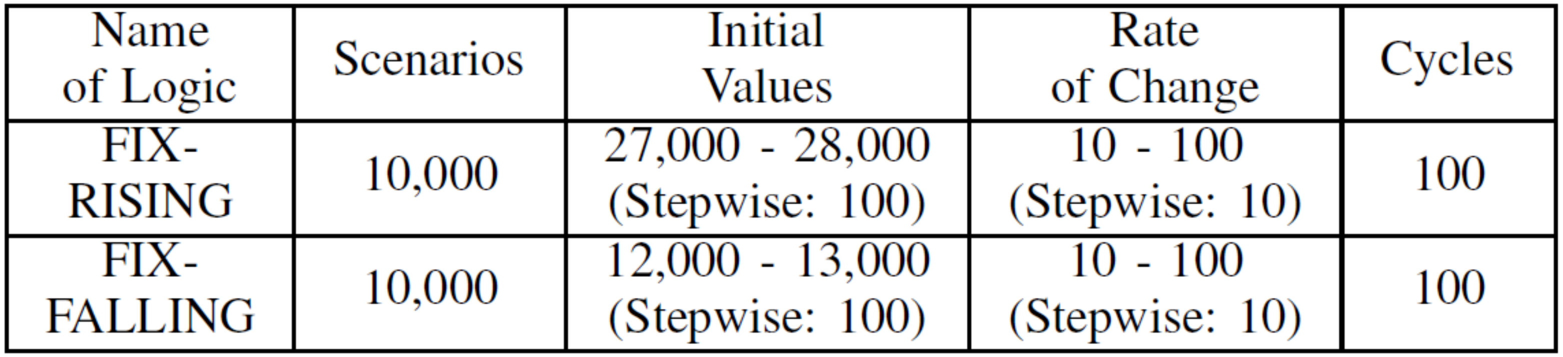
```
Model Input : BP.xml
Selection : FIX_RISING
ADD : 2
ADD : 3
Generation Start
Generation OK
```

```
Model Input : BP.xml
Selection : FIX_RISING
Simulation Start
Simulation result 생성
C:\Users\Sub\Desktop\a.txt
```

```
Comparision Start
Scenario_1_0 is OK
Comparision OK
```

Co-Simulation Environment

4.2 The Correctness Demonstration (G4)



Name of Logic	Scenarios	Initial Values	Rate of Change	Cycles
FIX-RISING	10,000	27,000 - 28,000 (Stepwise: 100)	10 - 100 (Stepwise: 10)	100
FIX-FALLING	10,000	12,000 - 13,000 (Stepwise: 100)	10 - 100 (Stepwise: 10)	100

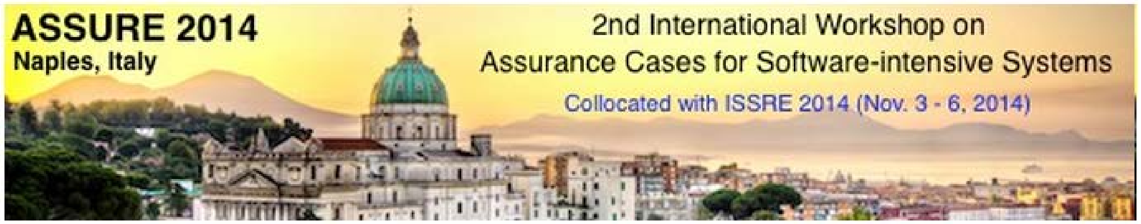
5. CONCLUSION AND FUTURE WORK

5. Conclusion

- This paper proposed an **indirect strategy for demonstrating the safety and correctness** of the '*FBDtoVerilog*' translator.
- We used the **safety case technique and GSN** to explain the proposed strategy more precisely and systematically.
- We also **developed several CASE tools** to support for deriving evidences.
  - '*FBDtoCadenceSMV*,' '*Scenario Generator*,' '*FBD Simulator*' and '*FBD-Verilog Comparator*.'
- We then **performed a case** study with an FBD program of the **KNICS APR-1400 RPS BP** in order to demonstrate the safety and correctness of the '*FBDtoVerilog*,' indirectly, according to the demonstration strategy proposed.

Future work

- We are now trying to **increase the confidence and thoroughness of the** '*Scenario Generator*'.
- We are also planning to **apply to other translators** which we developed, such as '*FBDtoC*' and '*NuSCRtoFBD*'.
- We expect to extend the proposed techniques into a safety and correctness demonstration framework for general translators and compilers.


ASSURE 2014
Naples, Italy
2nd International Workshop on
Assurance Cases for Software-intensive Systems
Collocated with ISSRE 2014 (Nov. 3 - 6, 2014)

Thank you for your attention …

Contact : atang34@konkuk.ac.kr (Eui-Sub Kim)