Generación automática de invariantes para implementar eficientemente Regiones Críticas Condicionales

Damián Barsotti

Universidad Nacional de Córdoba
Facultad de Matemática Astronomía y Física
damian@famaf.unc.edu.ar

26 de octubre de 2006

Resumen

1. Introducción
   - Marco general del trabajo
2. Representación de programas con lógica de primer orden
   - Predicados globales sobre programas
   - Corrección de programas (forma usual)
   - Transiciones
   - Sistema de transiciones
3. Optimización de regiones críticas condicionales
   - Regiones críticas condicionales
   - Semáforos Binarios Divididos (SBD)
4. Conclusión
   - Resultados y trabajos futuros

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP)
  - Solución predominante: uso de threads en programas
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP)
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Verificación de programas de forma automática.

- Corrección sobre programas ya terminados.
- Es indecidible para el caso general.
- Se prueba lo que se puede.

Este trabajo: Refinamiento automático (optimización).

- Nueva explosión de problemas concurrentes:
  - Masificación de arquitecturas paralelas (multicore processors, SMP).
  - Solución predominante: uso de threads en programas.
  - Difícil de programar, muchos errores.
- Implementar nuevas construcciones de más alto nivel.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Probadores de teoremas para programas.

Construcciones concurrentes de alto nivel

- Ej: patterns, monitores con automatic signalling, secciones, críticas condicionales, etc.
- Son ineficientes.
- Usar probadores de teoremas para optimizar (en compiladores).
- No importa (tanto) si es indecidible.

Probadores de teoremas.

- Primer orden y sat-solvers: CVC Lite, Vampire, ...
  - Por lo general totalmente automáticos.
  - Difícil modelar problemas en lógica.
- Alto orden: Isabelle, PVS, Coq, HOL, ...
  - Poco automáticos.
  - Orientados a modelar problemas.
- Usamos principalmente los primeros.

Modelo de programas en primer orden.

¿ Como transformar anotaciones a primer orden ?

Ejemplo (programa anotado):

$\{a = A \wedge b = B\}$
$a, b := a - b, a$
$\{a = A - B \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
$\quad \text{skip}$
$\Box\, a <= 0 \rightarrow$
$\quad a := -a$
$\underline{\text{fi}}$
$\{a = |A - B| \wedge b = A\}$

- Son locales,
- podría dar un predicado global.

Modelo de programas en primer orden.

¿ Como transformar anotaciones a primer orden ?

Ejemplo (programa anotado):

$\{a = A \wedge b = B\}$
$a, b := a - b, a$
$\{a = A - B \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
$\quad \text{skip}$
$\Box\, a <= 0 \rightarrow$
$\quad a := -a$
$\underline{\text{fi}}$
$\{a = |A - B| \wedge b = A\}$

- Son locales,
- podría dar un predicado global.

Modelo de programas en primer orden.

¿ Como transformar anotaciones a primer orden ?

Ejemplo (programa anotado):

$$
\begin{array}{l}
\{a = A \wedge b = B\} \\
\quad a, b := a - b, a \\
\{a = A - B \wedge b = A\} \\
\quad \underline{\text{if}}\ a >= 0 \rightarrow \\
\qquad \text{skip} \\
\quad \Box\, a <= 0 \rightarrow \\
\qquad a := -a \\
\quad \underline{\text{fi}} \\
\{a = |A - B| \wedge b = A\}
\end{array}
$$

- Son locales,
- podría dar un predicado global.

Modelo de programas en primer orden.

¿ Como transformar anotaciones a primer orden ?

Ejemplo (programa anotado):

0: $\{a = A \wedge b = B\}$
$a, b := a - b, a$
1: $\{a = A - B \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{a = |A - B| \wedge b = A\}$

- Son locales,
- podría dar un predicado global.
- Agreguemos "locaciones" (program counters).
- Predicado **global** sobre el programa:
$$\varphi : vc = 0 \Rightarrow a = A \wedge b = B \wedge \\ vc = 1 \Rightarrow a = A - B \wedge b = A \wedge \\ vc = 2 \Rightarrow a = |A - B| \wedge b = A$$
- ... si estoy en $vc = i$ entonces ...

Modelo de programas en primer orden.

¿ Como transformar anotaciones a primer orden ?

Ejemplo (programa anotado):

$0\colon \{a = A \wedge b = B\}$
$\quad a, b := a - b, a$
$1\colon \{a = A - B \wedge b = A\}$
$\quad \underline{\text{if}}\ a >= 0 \rightarrow$
$\qquad \text{skip}$
$\quad \square\, a <= 0 \rightarrow$
$\qquad a := -a$
$\quad \underline{\text{fi}}$
$2\colon \{a = |A - B| \wedge b = A\}$

- Son locales,
- podría dar un predicado global.
- Agreguemos "locaciones" (program counters).
- Predicado **global** sobre el programa:
  $\varphi : vc = 0 \Rightarrow a = A \wedge b = B \wedge$
  $vc = 1 \Rightarrow a = A - B \wedge b = A \wedge$
  $vc = 2 \Rightarrow a = |A - B| \wedge b = A$
- ... si estoy en $vc = i$ entonces ...

Modelo de programas en primer orden.

¿ Como transformar anotaciones a primer orden ?

Ejemplo (programa anotado):

0: $\{a = A \wedge b = B\}$
  $a, b := a - b, a$
1: $\{a = A - B \wedge b = A\}$
  $\underline{\text{if}}\ a >= 0 \rightarrow$
    skip
  $\Box\, a <= 0 \rightarrow$
    $a := -a$
  $\underline{\text{fi}}$
2: $\{a = |A - B| \wedge b = A\}$

- Son locales,
- podría dar un predicado global.
- Agreguemos "locaciones" (program counters).
- Predicado **global** sobre el programa:
  $$\begin{aligned}\varphi :\ & vc = 0 \Rightarrow a = A \wedge b = B \wedge \\ & vc = 1 \Rightarrow a = A - B \wedge b = A \wedge \\ & vc = 2 \Rightarrow a = |A - B| \wedge b = A\end{aligned}$$
- ... si estoy en $vc = i$ entonces ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0:
$a, b := a - b, a$
1:
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0:
$a, b := a - b, a$
1:
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\ a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0:
$a, b := a - b, a$
1:
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\square\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0:
$a, b := a - b, a$
1:
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\ a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0:
$a, b := a - b, a$
1: $\{a \geq 0 \Rightarrow \xi \wedge$
$a \leq 0 \Rightarrow \mathrm{wp} \,.\, a := -a \,.\, \xi\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\square\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0:
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0: $\{\mathrm{wp}\,.a, b := a - b, a.$
$(|a| = |A - B| \wedge b = A)\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\mathrm{if}}\ a >= 0 \rightarrow$
skip
$\Box\ a <= 0 \rightarrow$
$a := -a$
$\underline{\mathrm{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0: $\{ a = A \wedge (b = B \vee b = 2A - B)\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0: $\{ a = A \wedge$
$(b = B \vee b = 2A - B)\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\square\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...
- $\Theta \Rightarrow a = A \wedge (b = B \vee b = 2A - B)$ entonces es correcto.

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0: $\{\ a = A \wedge$
$(b = B \vee b = 2A - B)\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...
- $\Theta \Rightarrow a = A \wedge (b = B \vee b = 2A - B)$ entonces es correcto.
- Generalicemos wp a predicados globales,
- pero antes modelemos los programas con lógica.

Corrección de programas.

Θ : precondición
ξ : postcondición

$\{\Theta : a = A \wedge b = B\}$
0: $\{ a = A \wedge (b = B \vee b = 2A - B)\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\, a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- Si comienzo el programa cumpliendo Θ, si termina lo hace en ξ.
- Usemos wp para probarlo ...
- $\Theta \Rightarrow a = A \wedge (b = B \vee b = 2A - B)$ entonces es correcto.
- Generalicemos wp a predicados globales,
- pero antes modelemos los programas con lógica.

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

$$
\begin{array}{ll}
0: & \{\Theta : a = A \wedge b = B\} \\
 & a, b := a - b, a \\
1: & \underline{\text{if}}\ a >= 0 \rightarrow \\
 & \quad \text{skip} \\
 & \square\, a <= 0 \rightarrow \\
 & \quad a := -a \\
 & \underline{\text{fi}} \\
2: & \{\xi : a = |A - B| \wedge b = A\}
\end{array}
$$

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

$$
\begin{array}{ll}
0\text{:} & \{\Theta : a = A \wedge b = B\} \\
 & a, b := a - b, a \\
1\text{:} & \underline{\text{if}}\ a >= 0 \rightarrow \\
 & \quad \text{skip} \\
 & \square\, a <= 0 \rightarrow \\
 & \quad a := -a \\
 & \underline{\text{fi}} \\
2\text{:} & \{\xi : a = |A - B| \wedge b = A\}
\end{array}
$$

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

0: $\{\Theta : a = A \wedge b = B\}$
$\quad a, b := a - b, a$
1: $\quad \underline{\text{if}}\ a >= 0 \rightarrow$
$\qquad \text{skip}$
$\quad \Box\, a <= 0 \rightarrow$
$\qquad a := -a$
$\quad \underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

$$
\begin{array}{ll}
0: & \{\Theta : a = A \wedge b = B\} \\
 & a, b := a - b, a \\
1: & \underline{\text{if}}\ a >= 0 \rightarrow \\
 & \quad \text{skip} \\
 & \Box\, a <= 0 \rightarrow \\
 & \quad a := -a \\
 & \underline{\text{fi}} \\
2: & \{\xi : a = |A - B| \wedge b = A\}
\end{array}
$$

- transición de 0 a 1:
  $\Phi_{\tau_0} : a' = a - b \wedge b' = a$
- transición de 1 a 2 por $a \geq 0$:
  $\Phi_{\tau_1} : a \geq 0 \wedge a' = a \wedge b' = b$
- transición de 1 a 2 por $a \leq 0$:
  $\Phi_{\tau_2} : a \leq 0 \wedge a' = -a \wedge b' = b$

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

0: $\{\Theta : a = A \wedge b = B\}$
  $a, b := a - b, a$
1: $\underline{\text{if}}$ $a >= 0 \rightarrow$
  skip
  $\Box\, a <= 0 \rightarrow$
  $a := -a$
  $\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- transición de 0 a 1:
  $\Phi_{\tau_0} : a' = a - b \wedge b' = a$
- transición de 1 a 2 por $a \geq 0$:
  $\Phi_{\tau_1} : a \geq 0 \wedge a' = a \wedge b' = b$
- transición de 1 a 2 por $a \leq 0$:
  $\Phi_{\tau_2} : a \leq 0 \wedge a' = -a \wedge b' = b$

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

0: $\{\Theta : a = A \wedge b = B\}$
  $a, b := a - b, a$
1: $\underline{\text{if}}$ $a >= 0 \rightarrow$
    skip
  $\Box\, a <= 0 \rightarrow$
    $a := -a$
  $\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- transición de 0 a 1:
  $\Phi_{\tau_0} : a' = a - b \wedge b' = a$
- transición de 1 a 2 por $a \geq 0$:
  $\Phi_{\tau_1} : a \geq 0 \wedge a' = a \wedge b' = b$
- transición de 1 a 2 por $a \leq 0$:
  $\Phi_{\tau_2} : a \leq 0 \wedge a' = -a \wedge b' = b$

Predicados de transición.

- Acciones entre locaciones:
  - se componen de una asignación (posiblemente) guardada.
- Relación entre las variables antes y después:
  - variables primadas denotan el estado después del cambio.

0: $\{\Theta : a = A \wedge b = B\}$
    $a, b := a - b, a$
1: $\underline{\text{if}}\ a >= 0 \rightarrow$
        skip
    $\Box\, a <= 0 \rightarrow$
        $a := -a$
    $\underline{\text{fi}}$
2: $\{\xi : a = |A - B| \wedge b = A\}$

- transición de 0 a 1:
  $\Phi_{\tau_0} : a' = a - b \wedge b' = a$
- transición de 1 a 2 por $a \geq 0$:
  $\Phi_{\tau_1} : a \geq 0 \wedge a' = a \wedge b' = b$
- transición de 1 a 2 por $a \leq 0$:
  $\Phi_{\tau_2} : a \leq 0 \wedge a' = -a \wedge b' = b$

wp de una transición.

Definition

$$\mathrm{wp}\,.\,\Phi_\tau(\bar{x},\bar{x}').\,\varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x}' : \Phi_\tau(\bar{x},\bar{x}') : \varphi(\bar{x}')\rangle$$

Example (wp de Φ_{τ_2})

$$\begin{array}{ll}
 & \mathrm{wp}\,.\,\Phi_{\tau_2}(a,b,a'b').\,\varphi(a,b) \\
\equiv & \{\ \text{definición}\ \} \\
 & \langle \forall a', b' : a \le 0 \wedge a' = -a \wedge b' = b : \varphi(a',b')\rangle \\
\equiv & \{\ \text{intercambio}\ \} \\
 & \langle \forall a', b' : a' = -a \wedge b' = b : a \le 0 \Rightarrow \varphi(a',b')\rangle \\
\equiv & \{\ \text{rango unitario}\ \} \\
 & a \le 0 \Rightarrow \varphi(-a,b) \\
\equiv & \{\ \textbf{definición de wp usual}\ \} \\
 & a \le 0 \Rightarrow \mathrm{wp}\,.\,(a := -a)\,.\,\varphi
\end{array}$$

- Pero cada Φ_{τ_i} modela solo una transición...

wp de una transición.

Definition

$$\mathrm{wp}\,.\,\Phi_\tau(\bar{x},\bar{x}').\,\varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x}' : \Phi_\tau(\bar{x},\bar{x}') : \varphi(\bar{x}') \rangle$$

Example (wp de Φ_{τ_2})

$$\begin{aligned}
& \mathrm{wp}\,.\,\Phi_{\tau_2}(a,b,a'b').\,\varphi(a,b)\\
\equiv\;& \{\text{ definición }\}\\
& \langle \forall a',b' : a \le 0 \wedge a' = -a \wedge b' = b : \varphi(a',b') \rangle\\
\equiv\;& \{\text{ intercambio }\}\\
& \langle \forall a',b' : a' = -a \wedge b' = b : a \le 0 \Rightarrow \varphi(a',b') \rangle\\
\equiv\;& \{\text{ rango unitario }\}\\
& a \le 0 \Rightarrow \varphi(-a,b)\\
\equiv\;& \{\text{ \textbf{definición de wp usual} }\}\\
& a \le 0 \Rightarrow \mathrm{wp}\,.\,(a := -a).\,\varphi
\end{aligned}$$

- Pero cada Φ_{τ_i} modela solo una transición...

wp de una transición.

Definition

$$\mathrm{wp} \,.\, \Phi_\tau(\bar{x}, \bar{x'}) \,.\, \varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x'} : \Phi_\tau(\bar{x}, \bar{x'}) : \varphi(\bar{x'}) \rangle$$

Example (wp de Φ_{τ_2})

$$\begin{aligned}
& \mathrm{wp} \,.\, \Phi_{\tau_2}(a, b, a' b') \,.\, \varphi(a, b) \\
\equiv\; & \{ \text{ definición } \} \\
& \langle \forall a', b' : a \leq 0 \wedge a' = -a \wedge b' = b : \varphi(a', b') \rangle \\
\equiv\; & \{ \text{ intercambio } \} \\
& \langle \forall a', b' : a' = -a \wedge b' = b : a \leq 0 \Rightarrow \varphi(a', b') \rangle \\
\equiv\; & \{ \text{ rango unitario } \} \\
& a \leq 0 \Rightarrow \varphi(-a, b) \\
\equiv\; & \{ \textbf{ definición de wp usual } \} \\
& a \leq 0 \Rightarrow \mathrm{wp} \,.\, (a := -a) \,.\, \varphi
\end{aligned}$$

- Pero cada Φ_{τ_i} modela solo una transición...

wp de una transición.

Definition

$$\mathrm{wp}\,.\,\Phi_\tau(\bar{x},\bar{x}').\,\varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x}' : \Phi_\tau(\bar{x},\bar{x}') : \varphi(\bar{x}') \rangle$$

Example (wp de Φ_{τ_2})

$$\begin{aligned}
& \mathrm{wp}\,.\,\Phi_{\tau_2}(a,b,a'b').\,\varphi(a,b) \\
\equiv\; & \{\text{ definición }\} \\
& \langle \forall a', b' : a \leq 0 \wedge a' = -a \wedge b' = b : \varphi(a',b') \rangle \\
\equiv\; & \{\text{ intercambio }\} \\
& \langle \forall a', b' : a' = -a \wedge b' = b : a \leq 0 \Rightarrow \varphi(a',b') \rangle \\
\equiv\; & \{\text{ rango unitario }\} \\
& a \leq 0 \Rightarrow \varphi(-a,b) \\
\equiv\; & \{\text{ definición de wp usual }\} \\
& a \leq 0 \Rightarrow \mathrm{wp}\,.\,(a := -a).\,\varphi
\end{aligned}$$

- Pero cada Φ_{τ_i} modela solo una transición...

wp de una transición.

Definition

$$\mathrm{wp}\,.\,\Phi_\tau(\bar{x},\bar{x}').\,\varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x}' : \Phi_\tau(\bar{x},\bar{x}') : \varphi(\bar{x}') \rangle$$

Example (wp de Φ_{τ_2})

$$\begin{array}{rl} & \mathrm{wp}\,.\,\Phi_{\tau_2}(a,b,a'b').\,\varphi(a,b) \\ \equiv & \{\text{ definición }\} \\ & \langle \forall a',b' : a \leq 0 \wedge a' = -a \wedge b' = b : \varphi(a',b') \rangle \\ \equiv & \{\text{ intercambio }\} \\ & \langle \forall a',b' : a' = -a \wedge b' = b : a \leq 0 \Rightarrow \varphi(a',b') \rangle \\ \equiv & \{\text{ rango unitario }\} \\ & a \leq 0 \Rightarrow \varphi(-a,b) \\ \equiv & \{\textbf{ definición de wp usual }\} \\ & a \leq 0 \Rightarrow \mathrm{wp}\,.\,(a := -a).\,\varphi \end{array}$$

- Pero cada Φ_{τ_i} modela solo una transición...

wp de una transición.

Definition

$$\mathrm{wp}\,.\,\Phi_\tau(\bar{x}, \bar{x'}).\,\varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x'} : \Phi_\tau(\bar{x}, \bar{x'}) : \varphi(\bar{x'}) \rangle$$

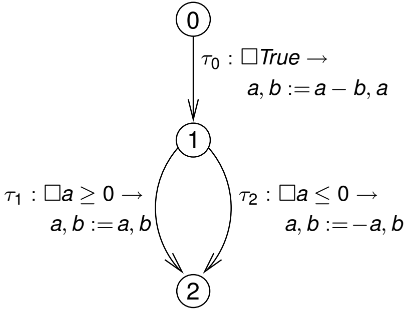
Example (wp de Φ_{τ_2})

$$
\begin{array}{ll}
 & \mathrm{wp}\,.\,\Phi_{\tau_2}(a, b, a' b').\,\varphi(a, b) \\
\equiv & \{ \text{ definición } \} \\
 & \langle \forall a', b' : a \leq 0 \wedge a' = -a \wedge b' = b : \varphi(a', b') \rangle \\
\equiv & \{ \text{ intercambio } \} \\
 & \langle \forall a', b' : a' = -a \wedge b' = b : a \leq 0 \Rightarrow \varphi(a', b') \rangle \\
\equiv & \{ \text{ rango unitario } \} \\
 & a \leq 0 \Rightarrow \varphi(-a, b) \\
\equiv & \{ \textbf{ definición de wp usual } \} \\
 & a \leq 0 \Rightarrow \mathrm{wp}\,.\,(a := -a).\,\varphi
\end{array}
$$

- Pero cada Φ_{τ_i} modela solo una transición...

Sistema de transiciones guardadas.

0:
$\quad a, b := a - b, a$
1:
$\quad \underline{\text{if}}\ a >= 0 \rightarrow$
$\qquad \text{skip}$
$\quad \square\ a <= 0 \rightarrow$
$\qquad a := -a$
$\quad \underline{\text{fi}}$
2:



Definition (Grafo de transiciones)

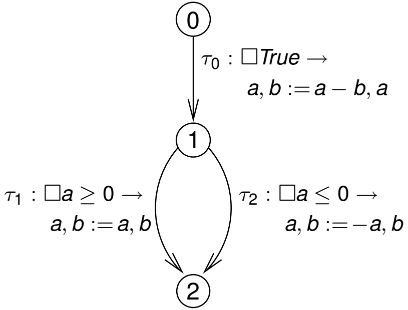
$\mathcal{L} = \{0, \cdots, m\}$ locaciones. $\quad \mathcal{T} = \{\tau_0, \cdots, \tau_m\}$ transiciones.

s_{τ_i} locación de salida. $\quad \gamma_{\tau_i}(\bar{x})$ guarda.

t_{τ_i} locación de entrada. $\quad \bar{x} := \bar{e}_{\tau_i}(\bar{x})$ asignación múltiple.

Sistema de transiciones guardadas.

0:
$a, b := a - b, a$
1:
$\underline{\text{if}}\ a >= 0 \rightarrow$
skip
$\Box\ a <= 0 \rightarrow$
$a := -a$
$\underline{\text{fi}}$
2:



Definition (Grafo de transiciones)

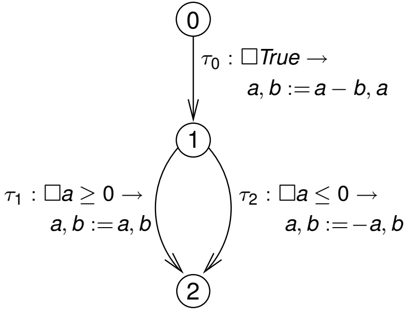
$\mathcal{L} = \{0, \cdots, m\}$ locaciones. $\mathcal{T} = \{\tau_0, \cdots, \tau_m\}$ transiciones.

s_{τ_i} locación de salida. $\gamma_{\tau_i}(\bar{x})$ guarda.

t_{τ_i} locación de entrada. $\bar{x} := \bar{e}_{\tau_i}(\bar{x})$ asignación múltiple.

Sistema de transiciones guardadas.

0:
$\quad a, b := a - b, a$
1:
$\quad \underline{\text{if}}\ a >= 0 \rightarrow$
$\qquad \text{skip}$
$\quad \Box\ a <= 0 \rightarrow$
$\qquad a := -a$
$\quad \underline{\text{fi}}$
2:



Definition (Grafo de transiciones)

$\mathcal{L} = \{0, \cdots, m\}$ locaciones. $\quad \mathcal{T} = \{\tau_0, \cdots, \tau_m\}$ transiciones.

s_{τ_i} locación de salida. $\quad \gamma_{\tau_i}(\bar{x})$ guarda.

t_{τ_i} locación de entrada. $\quad \bar{x} := \bar{e}_{\tau_i}(\bar{x})$ asignación múltiple.

Sistema de transiciones guardadas.

0:

$\quad a, b := a - b, a$

1:

$\quad \underline{\text{if}}\ a >= 0 \rightarrow$

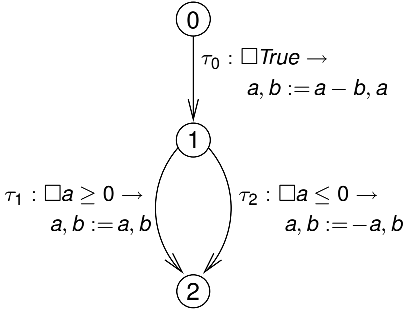
$\qquad \text{skip}$

$\quad \Box\, a <= 0 \rightarrow$

$\qquad a := -a$

$\quad \underline{\text{fi}}$

2:



Definition (Grafo de transiciones)

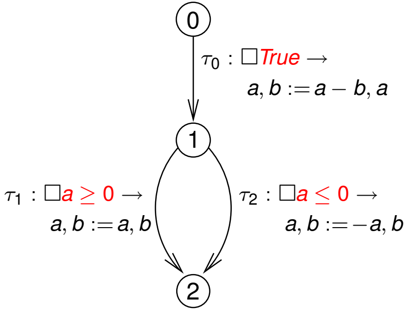
$\mathcal{L} = \{0, \cdots, m\}$ locaciones. $\mathcal{T} = \{\tau_0, \cdots, \tau_m\}$ transiciones.

s_{τ_i} locación de salida. $\gamma_{\tau_i}(\bar{x})$ guarda.

t_{τ_i} locación de entrada. $\bar{x} := \bar{e}_{\tau_i}(\bar{x})$ asignación múltiple.

Sistema de transiciones guardadas.

0:
$\quad a, b := a - b, a$
1:
$\quad \underline{\text{if}}\ a >= 0 \rightarrow$
$\qquad \text{skip}$
$\quad \Box\, a <= 0 \rightarrow$
$\qquad a := -a$
$\quad \underline{\text{fi}}$
2:



Definition (Grafo de transiciones)

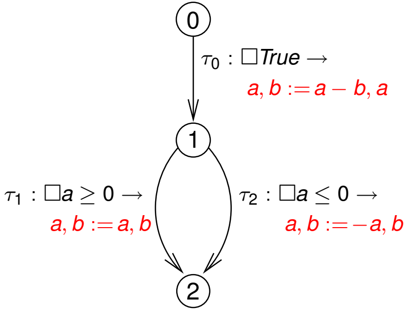
$\mathcal{L} = \{0, \cdots, m\}$ locaciones. $\mathcal{T} = \{\tau_0, \cdots, \tau_m\}$ transiciones.

s_{τ_i} locación de salida. $\gamma_{\tau_i}(\bar{x})$ guarda.

t_{τ_i} locación de entrada. $\bar{x} := \bar{e}_{\tau_i}(\bar{x})$ asignación múltiple.

Sistema de transiciones guardadas.

0:
  $a, b := a - b, a$
1:
  $\underline{\text{if}}\ a >= 0 \rightarrow$
    skip
  $\Box\ a <= 0 \rightarrow$
    $a := -a$
  $\underline{\text{fi}}$
2:



Definition (Grafo de transiciones)

$\mathcal{L} = \{0, \cdots, m\}$ locaciones. $\mathcal{T} = \{\tau_0, \cdots, \tau_m\}$ transiciones.

s_{τ_i} locación de salida. $\gamma_{\tau_i}(\bar{x})$ guarda.

t_{τ_i} locación de entrada. $\bar{x} := \bar{e}_{\tau_i}(\bar{x})$ asignación múltiple.

WP de un sistema de transiciones.

Definition (Predicado de sistema)

$$\Phi(\bar{x}, vc, \bar{x}', vc') \;\equiv\; \bigvee_{\tau \in \mathcal{T}} vc = s_\tau \wedge \Phi_\tau(\bar{x}, \bar{x}') \wedge vc' = t_\tau$$

Definition (WP de un sistema de transiciones)

$$\mathrm{WP} \,.\, \Phi(\bar{x}, vc, \bar{x}', vc') . \varphi(\bar{x}) \;\equiv\; \langle \forall \bar{x}', vc' : \Phi(\bar{x}, vc, \bar{x}', vc') : \varphi(\bar{x}') \rangle$$

Theorem

$$WP \,.\, \Phi(\bar{x}, vc, \bar{x}', vc') . \varphi(\bar{x}) \;\equiv\; \bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \bigwedge_{\substack{\tau \in \mathcal{T} \wedge \\ s_\tau = i}} wp \,.\, \Phi_\tau . \varphi_i(\bar{x})$$

Corrección.

Definition (arreglos)

$$\bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \varphi_i \equiv [\varphi_0, \cdots, \varphi_m]$$

$$\Theta = [a = A \wedge b = B, False, False]$$
$$\xi = [True, True, a = |A - B| \wedge b = A]$$

0:
  $a, b := a - b, a$
1:
  if $a >= 0 \rightarrow$ skip
  $\square\, a <= 0 \rightarrow a := -a$
  fi
2:

Calculemos con WP . Φ:

$\varphi_1 = \xi$
$\varphi_2 = [True, |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$
$\varphi_3 = [a = A \wedge (b = B \vee b = 2A - B), |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$
$\varphi_4 = \varphi_3$

Cálculo de punto fijo $\varphi_{i+1} = \xi \wedge \mathrm{WP} . \Phi . \varphi_i$ con $\varphi_0 = [True, True, True]$

Corrección.

Definition (arreglos)

$$\bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \varphi_i \equiv [\varphi_0, \cdots, \varphi_m]$$

0: $\{True\}$
$a, b := a - b, a$
1: $\{True\}$
$\underline{\text{if}}\ a >= 0 \rightarrow \text{skip}$
$\Box\ a <= 0 \rightarrow a := -a$
$\underline{\text{fi}}$
2: $\{a = |A - B| \wedge b = A\}$

$\Theta = [a = A \wedge b = B, False, False]$

$\xi = [True, True, a = |A - B| \wedge b = A]$

Calculemos con $\mathrm{WP}\,.\,\Phi$:

$\varphi_1 = \xi$

$\varphi_2 = [True, |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$

$\varphi_3 = [a = A \wedge (b = B \vee b = 2A - B), |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$

$\varphi_4 = \varphi_3$

Cálculo de punto fijo $\varphi_{i+1} = \xi \wedge \mathrm{WP}\,.\,\Phi\,.\,\varphi_i$ con $\varphi_0 = [True, True, True]$

Corrección.

Definition (arreglos)

$$\bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \varphi_i \equiv [\varphi_0, \cdots, \varphi_m]$$

$\Theta = [a = A \wedge b = B, False, False]$

$\xi = [True, True, a = |A - B| \wedge b = A]$

0: $\{True\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow \text{skip}$
$\Box\ a <= 0 \rightarrow a := -a$
$\underline{\text{fi}}$
2: $\{a = |A - B| \wedge b = A\}$

Calculemos con $\mathrm{WP}.\Phi$:

$\varphi_1 = \xi$

$\varphi_2 = [True, |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$

$\varphi_3 = [a = A \wedge (b = B \vee b = 2A - B), |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$

$\varphi_4 = \varphi_3$

Cálculo de punto fijo $\varphi_{i+1} = \xi \wedge \mathrm{WP}.\Phi.\varphi_i$ con $\varphi_0 = [True, True, True]$

Corrección.

Definition (arreglos)

$$\bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \varphi_i \quad \equiv \quad [\varphi_0, \cdots, \varphi_m]$$

0: $\{\, a = A \wedge (b = B \vee b = 2A - B)\}$
$a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\underline{\text{if}}\ a >= 0 \rightarrow \text{skip}$
$\square\ a <= 0 \rightarrow a := -a$
$\underline{\text{fi}}$
2: $\{a = |A - B| \wedge b = A\}$

$$\Theta = [a = A \wedge b = B, False, False]$$
$$\xi = [True, True, a = |A - B| \wedge b = A]$$

Calculemos con $\mathrm{WP} . \Phi$:

$$\varphi_1 = \xi$$
$$\varphi_2 = [True, |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$$
$$\varphi_3 = [a = A \wedge (b = B \vee b = 2A - B), |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$$
$$\varphi_4 = \varphi_3$$

Cálculo de punto fijo $\varphi_{i+1} = \xi \wedge \mathrm{WP} . \Phi . \varphi_i$ con $\varphi_0 = [True, True, True]$

Corrección.

Definition (arreglos)

$$\bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \varphi_i \quad \equiv \quad [\varphi_0, \cdots, \varphi_m]$$

0: $\{\, a = A \wedge (b = B \vee b = 2A - B)\}$
$\quad a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
$\quad$ <u>if</u> $a >= 0 \rightarrow$skip
$\quad \Box\, a <= 0 \rightarrow a := -a$
$\quad$ <u>fi</u>
2: $\{a = |A - B| \wedge b = A\}$

$$\Theta = [a = A \wedge b = B, False, False]$$
$$\xi = [True, True, a = |A - B| \wedge b = A]$$

Calculemos con $\mathrm{WP}\,.\,\Phi$:

$$\varphi_1 = \xi$$
$$\varphi_2 = [True, |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$$
$$\varphi_3 = [a = A \wedge (b = B \vee b = 2A - B), |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$$
$$\varphi_4 = \varphi_3$$

Cálculo de punto fijo $\varphi_{i+1} = \xi \wedge \mathrm{WP}\,.\,\Phi\,.\,\varphi_i$ con $\varphi_0 = [True, True, True]$

Corrección.

Definition (arreglos)

$$\bigwedge_{i \in \mathcal{L}} vc = i \Rightarrow \varphi_i \qquad \equiv \qquad [\varphi_0, \cdots, \varphi_m]$$

$$\Theta = [a = A \wedge b = B, False, False]$$
$$\xi = [True, True, a = |A - B| \wedge b = A]$$

0: $\{ a = A \wedge (b = B \vee b = 2A - B)\}$
   $a, b := a - b, a$
1: $\{|a| = |A - B| \wedge b = A\}$
   $\underline{\text{if}}\ a >= 0 \rightarrow \text{skip}$
   $\square\ a <= 0 \rightarrow a := -a$
   $\underline{\text{fi}}$
2: $\{a = |A - B| \wedge b = A\}$

Calculemos con $\mathrm{WP}\,.\,\Phi$:

$$\varphi_1 = \xi$$
$$\varphi_2 = [True, |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$$
$$\varphi_3 = [a = A \wedge (b = B \vee b = 2A - B), |a| = |A - B| \wedge b = A, a = |A - B| \wedge b = A]$$
$$\varphi_4 = \varphi_3$$

Cálculo de punto fijo $\varphi_{i+1} = \xi \wedge \mathrm{WP}\,.\,\Phi\,.\,\varphi_i$ con $\varphi_0 = [True, True, True]$

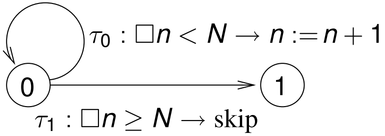
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP} \,.\, \phi .\, Y$$

$\Theta = [n < N, \mathit{False}]$
$\xi = [\mathit{True}, n = N]$

0: <u>while</u> $n < N$
    $n := n + 1;$
1:



$\varphi_0 = [\mathit{True}, \mathit{True}]$
$\varphi_1 = [\mathit{True}, n = N] \ (= \xi)$
$\varphi_2 = [n \leq N, n = N]$
$\varphi_3 = [n \leq N, n = N] \ (= \varphi_2)$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

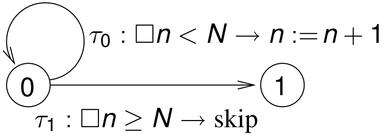
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP}\,.\,\phi.\,Y$$

$\Theta = [n < N, False]$
$\xi = [True, n = N]$

0: <u>while</u> $n < N$
  $n := n + 1;$
1:



$\varphi_0 = [True, True]$
$\varphi_1 = [True, n = N]\ (= \xi)$
$\varphi_2 = [n \leq N, n = N]$
$\varphi_3 = [n \leq N, n = N]\ (= \varphi_2)$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

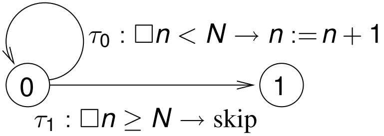
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP}\,.\,\phi.\, Y$$

$\Theta = [n < N, False]$
$\xi = [True, n = N]$

```
0: while n < N
       n := n + 1;
1:
```



$$\varphi_0 = [True, True]$$
$$\varphi_1 = [True, n = N] \ (= \xi)$$
$$\varphi_2 = [n \leq N, n = N]$$
$$\varphi_3 = [n \leq N, n = N] \ (= \varphi_2)$$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

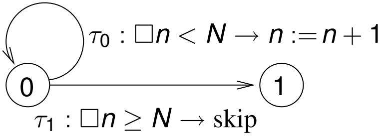
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP} \,.\, \phi .\ Y$$

$\Theta = [n < N, \mathit{False}]$
$\xi = [\mathit{True}, n = N]$

0: <u>while</u> $n < N$
    $n := n + 1;$
1:



$$\begin{aligned}
\varphi_0 &= [\mathit{True}, \mathit{True}] \\
\varphi_1 &= [\mathit{True}, n = N]\ (= \xi) \\
\varphi_2 &= [n \leq N, n = N] \\
\varphi_3 &= [n \leq N, n = N]\ (= \varphi_2)
\end{aligned}$$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

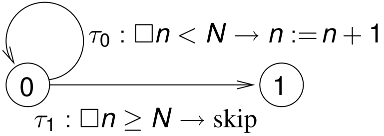
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP} \,.\, \phi .\, Y$$

$\Theta = [n < N, \mathit{False}]$
$\xi = [\mathit{True}, n = N]$

0: <u>while</u> $n < N$
    $n := n + 1;$
1:



$\varphi_0 = [\mathit{True}, \mathit{True}]$
$\varphi_1 = [\mathit{True}, n = N] \; (= \xi)$
$\varphi_2 = [n \leq N, n = N]$
$\varphi_3 = [n \leq N, n = N] \; (= \varphi_2)$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

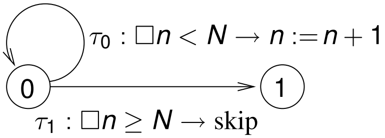
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP}\,.\,\phi.\, Y$$

$\Theta = [n < N, False]$
$\xi = [True, n = N]$

0: <u>while</u> $n < N$
 $n := n + 1;$
1:



$\varphi_0 = [True, True]$
$\varphi_1 = [True, n = N]\ (= \xi)$
$\varphi_2 = [n \leq N, n = N]$
$\varphi_3 = [n \leq N, n = N]\ (= \varphi_2)$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

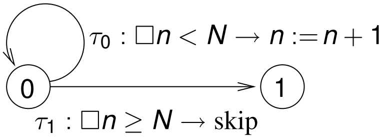
Programa iterativo.

Punto fijo del transformador

$$\mathcal{B}(Y) \triangleq \xi \wedge \mathrm{WP} \,.\, \phi .\, Y$$

$\Theta = [n < N, \mathit{False}]$
$\xi = [\mathit{True}, n = N]$

0: <u>while</u> $n < N$
$\quad n := n + 1;$
1:



$$\begin{aligned}
\varphi_0 &= [\mathit{True}, \mathit{True}] \\
\varphi_1 &= [\mathit{True}, n = N] \ (= \xi) \\
\varphi_2 &= [n \leq N, n = N] \\
\varphi_3 &= [n \leq N, n = N] \ (= \varphi_2)
\end{aligned}$$

- φ_2 es el mayor punto fijo de $\mathcal{B}$.
- $\Theta \Rightarrow \varphi_2$, entonces
- φ_2 es un invariante (inductivo).

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Problemas de regiones críticas condicionales.

Especificación:

- m programas $S_0, \cdots, S_{m-1}$ a ejecutarse en exclusión mutua condicional.
- Los programas acceden a un recurso compartido.
- m condiciones $B_0, \cdots, B_{m-1}$ de ejecución.
- Invariante global del sistema I.
- Múltiples procesos compiten por el recurso (ejecutando los programas).

Soluciones:

- Monitores.
- Semáforos binarios divididos.

Semáforos Binarios Divididos (SBD).

A partir de especificación se construyen m nuevos programas **automáticamente**.

Entrada $P.s_m$, $P.s_i$
S_i
Salida $V.s_i, \cdots, V.s_m$

- Se agregan $m+1$ semáforos binarios $\{s_0, \cdots, s_m\}$.
- A lo sumo uno está activado $SBS : 0 \leq \langle \Sigma i : 0 \leq i \leq m : s_i \rangle \leq 1$.
  - Solo un proceso en ejecución.
  - Asegura exclusión mutua.
- Regla del dominó: un V será seguido de un P sobre el mismo semáforo en otro proceso.

Esto permite modelarlos con sistemas de transiciones guardadas.

Semáforos Binarios Divididos (SBD).

A partir de especificación se construyen m nuevos programas **automáticamente**.

Entrada $P.\,s_m\ ,\ P.\,s_i$
S_i
Salida $V.\,s_i, \cdots, V.\,s_m$

- Se agregan $m + 1$ semáforos binarios $\{s_0, \cdots, s_m\}$.
- A lo sumo uno está activado $SBS : 0 \leq \langle \Sigma i : 0 \leq i \leq m : s_i \rangle \leq 1$.
  - Solo un proceso en ejecución.
  - Asegura exclusión mutua.
- Regla del dominó: un V será seguido de un P sobre el mismo semáforo en otro proceso.

Esto permite modelarlos con sistemas de transiciones guardadas.

Semáforos Binarios Divididos (SBD).

A partir de especificación se construyen m nuevos programas **automáticamente**.

Entrada $P \,.\, s_m \;,\; P \,.\, s_i$
S_i
Salida $V \,.\, s_i, \cdots, V \,.\, s_m$

- Se agregan $m + 1$ semáforos binarios $\{s_0, \cdots, s_m\}$.
- A lo sumo uno está activado $SBS : 0 \leq \langle \Sigma i : 0 \leq i \leq m : s_i \rangle \leq 1$.
  - Solo un proceso en ejecución.
  - Asegura exclusión mutua.
- Regla del dominó: un V será seguido de un P sobre el mismo semáforo en otro proceso.

Esto permite modelarlos con sistemas de transiciones guardadas.

Semáforos Binarios Divididos (SBD).

A partir de especificación se construyen m nuevos programas **automáticamente**.

Entrada $P\,.\,s_m\ ,\ P\,.\,s_i$
S_i
Salida $V\,.\,s_i,\cdots,V\,.\,s_m$

- Se agregan $m+1$ semáforos binarios $\{s_0,\cdots,s_m\}$.
- A lo sumo uno está activado $SBS : 0 \leq \langle \Sigma i : 0 \leq i \leq m : s_i \rangle \leq 1$.
  - Solo un proceso en ejecución.
  - Asegura exclusión mutua.
- Regla del dominó: un V será seguido de un P sobre el mismo semáforo en otro proceso.

Esto permite modelarlos con sistemas de transiciones guardadas.

Semáforos Binarios Divididos (SBD).

A partir de especificación se construyen m nuevos programas **automáticamente**.

<u>Entrada</u> $P.s_m$, $P.s_i$
S_i
<u>Salida</u> $V.s_i, \cdots, V.s_m$

- Se agregan $m+1$ semáforos binarios $\{s_0, \cdots, s_m\}$.
- A lo sumo uno está activado $SBS : 0 \leq \langle \Sigma i : 0 \leq i \leq m : s_i \rangle \leq 1$.
  - Solo un proceso en ejecución.
  - Asegura exclusión mutua.
- Regla del dominó: un V será seguido de un P sobre el mismo semáforo en otro proceso.

Esto permite modelarlos con sistemas de transiciones guardadas.

Productor Consumidor con Buffer Acotado (Ejemplo).

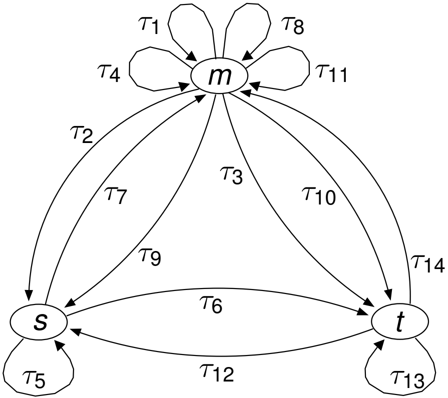
$$
\begin{array}{l}
\mathrm{P}.m; \\
\underline{\mathrm{if}}\quad n < N \rightarrow \mathrm{skip} \\
\Box\; n = N \rightarrow b := b + 1;\ \mathrm{V}.m\ ; \\
\qquad\qquad \mathrm{P}.s\ ;\ b := b - 1 \\
\underline{\mathrm{fi}}; \\
n := n + 1; \\
\underline{\mathrm{if}}\quad n < N \wedge 0 < b \rightarrow \mathrm{V}.s \\
\Box\; 0 < n \wedge 0 < c \rightarrow \mathrm{V}.t \\
\Box\; (n = N \vee 0 = b) \wedge (0 = n \vee 0 = c) \\
\qquad\qquad \rightarrow \mathrm{V}.m \\
\underline{\mathrm{fi}}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{P}.m; \\
\underline{\mathrm{if}}\quad 0 < n \rightarrow \mathrm{skip} \\
\Box\; 0 = n \rightarrow c := c + 1;\ \mathrm{V}.m\ ; \\
\qquad\qquad \mathrm{P}.t\ ;\ c := c - 1 \\
\underline{\mathrm{fi}}; \\
n := n - 1; \\
\underline{\mathrm{if}}\quad n < N \wedge 0 < b \rightarrow \mathrm{V}.s \\
\Box\; 0 < n \wedge 0 < c \rightarrow \mathrm{V}.t \\
\Box\; (n = N \vee 0 = b) \wedge (0 = n \vee 0 = c) \\
\qquad\qquad \rightarrow \mathrm{V}.m \\
\underline{\mathrm{fi}}
\end{array}
$$

Por propiedad SBS y regla del dominó:

- Modelamos cada traza como una transición.
- Las locaciones serán los semáforos $\mathcal{L} = \{s, t, m\}$.

Productor Consumidor con Buffer Acotado (Ejemplo).



Por propiedad SBS y regla del dominó:

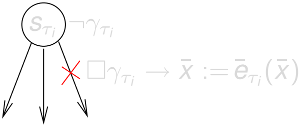
- Modelamos cada traza como una transición.
- Las locaciones serán los semáforos $\mathcal{L} = \{s, t, m\}$.

Simplificación de programas.

- Para la mayoría de los problemas hay guardas finales que nunca se ejecutan.
- Intentar probar corrección con ξ denotando la negación de una guarda:

$$\xi : vc = s_{\tau_i} \Rightarrow \neg\gamma_{\tau_i}$$

- Si converge podemos simplificar el programa

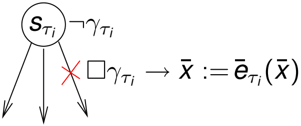


Simplificación de programas.

- Para la mayoría de los problemas hay guardas finales que nunca se ejecutan.
- Intentar probar corrección con ξ denotando la negación de una guarda:

$$\xi : vc = s_{\tau_i} \Rightarrow \neg\gamma_{\tau_i}$$

- Si converge podemos simplificar el programa

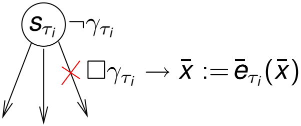


Simplificación de programas.

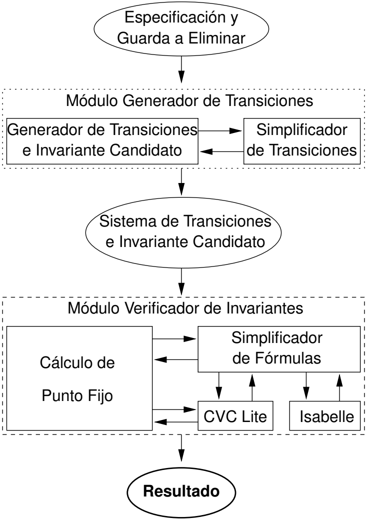
- Para la mayoría de los problemas hay guardas finales que nunca se ejecutan.
- Intentar probar corrección con ξ denotando la negación de una guarda:

$$\xi : vc = s_{\tau_i} \Rightarrow \neg\gamma_{\tau_i}$$

- Si converge podemos simplificar el programa

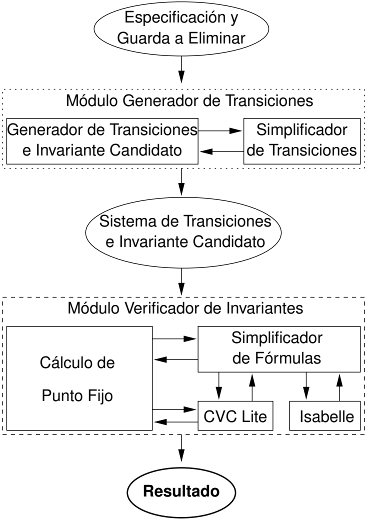


Software.



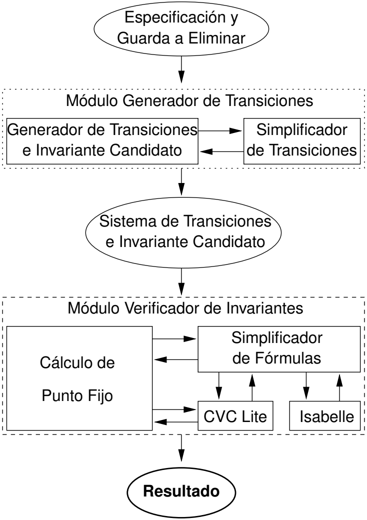
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



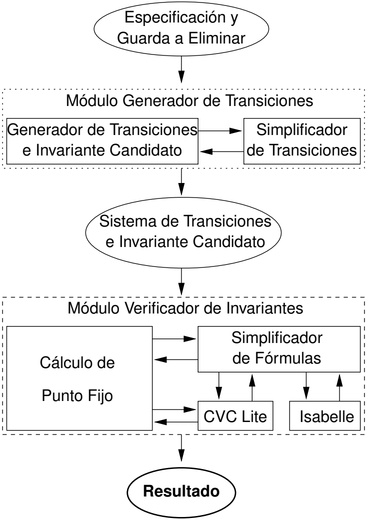
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



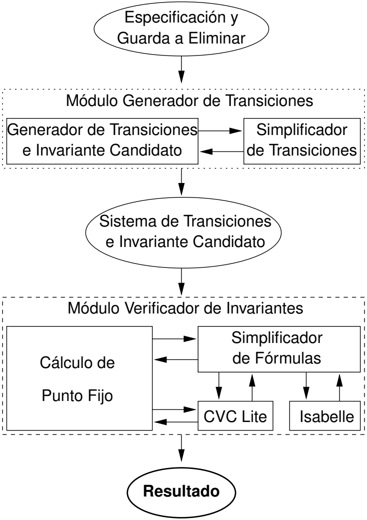
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



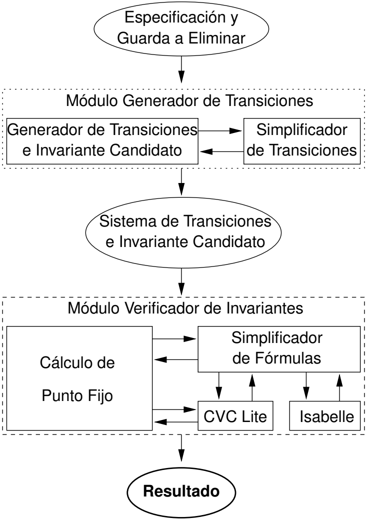
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



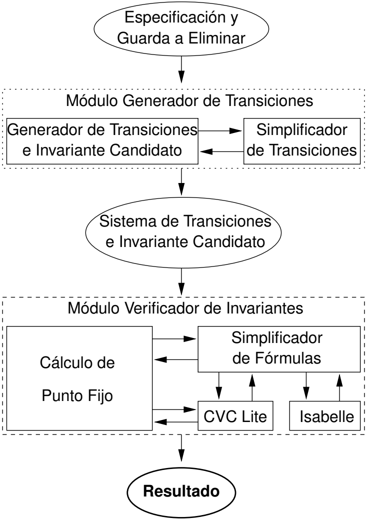
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



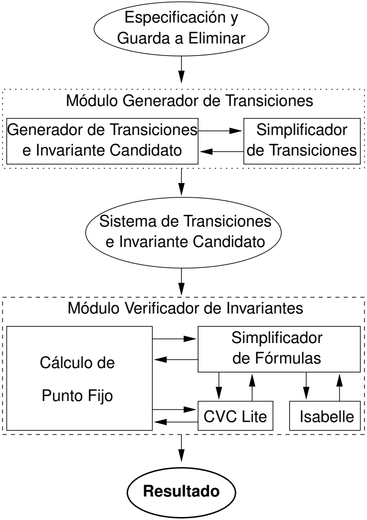
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



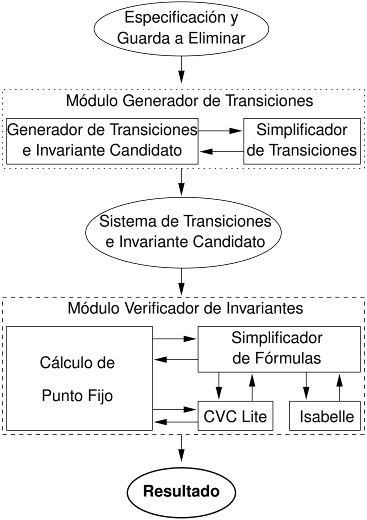
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



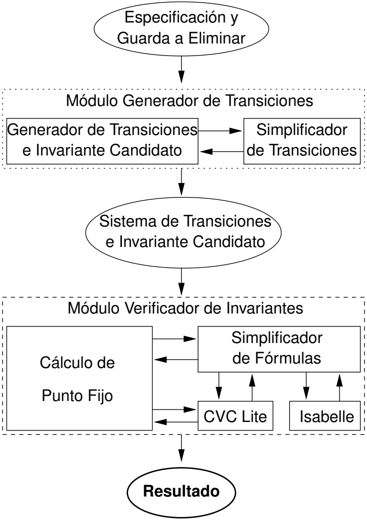
- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Software.



- Especificación de RCC y guarda.
- Generador de transiciones.
  - Produce transiciones y ξ.
  - Simplifica ambas (código ML).
- Verificador de Invariantes.
  - Busca el punto fijo (con máximo de iteraciones) y
  - detecta imposibilidad de convergencia,
  - probando implicaciones con CVC Lite.
  - Simplifica fórmulas,
  - utilizando tácticas propias con ML y CVC Lite, Isabelle.

Resultados.

Problemas:

- Semáforos Generales.
- Productor/Consumidor y Productor/Consumidor Goloso.
- Lectores y Escritores.
- Productor/Consumidor consumiendo de a n.
- Converge o detecta imposibilidad en pocos segundos,

Resultados.

Problemas:

- Semáforos Generales.
- Productor/Consumidor y Productor/Consumidor Goloso.
- Lectores y Escritores.
- Productor/Consumidor consumiendo de a *n*.
- Converge o detecta imposibilidad en pocos segundos,

Resultados.

Problemas:

- Semáforos Generales.
- Productor/Consumidor y Productor/Consumidor Goloso.
- Lectores y Escritores.
- Productor/Consumidor consumiendo de a *n*.
- Converge o detecta imposibilidad en pocos segundos,
- excepto para una guarda del último problema, pero ...
- hecho a mano tampoco converge.

Resultados.

Problemas:

- Semáforos Generales.
- Productor/Consumidor y Productor/Consumidor Goloso.
- Lectores y Escritores.
- Productor/Consumidor consumiendo de a *n*.
- Converge o detecta imposibilidad en pocos segundos,
- excepto para una guarda del último problema, pero ...
- hecho a mano tampoco converge.

Trabajos futuros (actuales):

- Abstract interpretation en el prototipo (poliedros convexos, ideal de polinomios, etc.).
- Implementación y optimización automática de monitores con automatic signalling en Java.

Resultados.

Problemas:

- Semáforos Generales.
- Productor/Consumidor y Productor/Consumidor Goloso.
- Lectores y Escritores.
- Productor/Consumidor consumiendo de a *n*.
- Converge o detecta imposibilidad en pocos segundos,
- excepto para una guarda del último problema, pero ...
- hecho a mano tampoco converge.

Trabajos futuros (actuales):

- Abstract interpretation en el prototipo (poliedros convexos, ideal de polinomios, etc.).
- Implementación y optimización automática de monitores con automatic signalling en Java.

Resultados.

Problemas:

- Semáforos Generales.
- Productor/Consumidor y Productor/Consumidor Goloso.
- Lectores y Escritores.
- Productor/Consumidor consumiendo de a n.
- Converge o detecta imposibilidad en pocos segundos,
- excepto para una guarda del último problema, pero ...
- hecho a mano tampoco converge.

Trabajos futuros (actuales):

- Abstract interpretation en el prototipo (poliedros convexos, ideal de polinomios, etc.).
- Implementación y optimización automática de monitores con automatic signalling en Java.

FIN.