



INSTITUTO TECNOLÓGICO DE IZTAPALAPA I

INGENIERIA EN SISTEMAS COMPUTACIONALES

Presenta el trabajo titulado

REPORTES DE APUNTES DEL SEMESTRE

MARZO-JULIO 2021

Presenta:

PEREZ ARMAS FAUSTO ISAAC

No. De control:

181080037

Asignatura:

LENGUAJE Y AUTOMATAS II

ASESOR INTERNO:

M.C. ABIEL TOMAS PARRA HERNANDEZ

CIUDAD DE MEXICO

MARZO-JUNIO/2021

INDICE

A) Actividades semana marzo 8-12, 2021	1
Actividades individuales	1
1) Conferencia: ¿Son las computadoras todopoderosas? de Sergio Rajsbaum	1
2) Golpe cibernético global - Alfredo Jalife Rahme	3
3) Test de personalidad.....	4
4) Collage digital de sus gustos y pasiones.....	6
B) Actividades semana marzo 16-19, 2021	7
Actividades individuales:	7
1) Película The Imitation Game.....	7
Actividades en equipo:.....	9
2) Ensayo / Resumen Untangling the Tale of Ada Lovelace de Stephen Wolfram.....	9
C) Actividades semana abril 12-16, 2021	11
Actividades individuales:	11
1) Artículo Translator (computing) de Wikipedia.	11
2) Artículo The Latin American Supercomputing Ecosystem for Science de la revista Communications de la ACM.	13
.....	13
Actividades en equipo:.....	14
3) Resumen de ambos artículos y grabar un video	14
Resumen del equipo	14
Resumen personal	19
IMÁGENES DEL VIDEO	21
LIGA DEL VIDEO:	24
D) Actividades semana abril 19-23, 2021.....	25
Actividades individuales	25
Introduction" y "Compiler Structure". Capítulo 1 del libro "Engineering a Compiler" de Cooper y Torczon (2012).	25
Actividades en equipo.....	31
2) "Getting Started / Tutorials" de la documentación oficial de LLVM.	31
E) Actividades semana abril 26-30, 2021	33
Actividades individuales:	33
1) "Overview of Translation" del capítulo 1 del libro "Engineering a Compiler" de Cooper y Torczon (2012).	33

Actividades en equipo:.....	38
2) Multi-Level Intermediate Representation (MLIR).....	38
• MLIR	38
F) Actividades semana mayo 3-7, 2021.....	39
Actividades individuales	39
1) "Chapter Summary and Perspective" del libro "Engineering a Compiler" de Cooper y Torczon (2012).	39
Actividades en equipo.....	46
3) Video de presentación de avance del proyecto final del concepto "Multi-Level Intermediate Representation (MLIR)".	46
LIGA DEL VIDEO:	49
G) Actividades en clase:	50
Capturas de los ejercicios en clase	50

A) Actividades semana marzo 8-12, 2021

Actividades individuales

1) Conferencia: ¿Son las computadoras todopoderosas? de Sergio Rajsbaum

Para empezar, Supercomputadora, supercomputador o superordenador es un dispositivo informático con capacidades de cálculo superiores a las computadoras comunes y de escritorio, ya que son usadas con fines específicos. Hoy día los términos de supercomputadora y superordenador están siendo reemplazados por computadora de alto rendimiento y ambiente de cómputo de alto rendimiento, ya que las supercomputadoras son un conjunto de poderosos ordenadores unidos entre sí para aumentar su potencia de trabajo y rendimiento. Al año 2019, los superordenadores más rápidos funcionaban en aproximadamente más de 148 petaflops (un petaflop, en la jerga de la computación, significa que realizan más de 1000 billones de operaciones por segundo).¹ La lista de supercomputadoras se encuentra en el ranking TOP500.

Sistemas de enfriamiento

Muchas de las CPUs usadas en los supercomputadores de hoy disipan 10 veces más calor que un disco de estufa común. Algunos diseños necesitan enfriar los múltiples CPUs a -85°C (-185°F).

Para poder enfriar múltiples CPUs a tales temperaturas requiere de un gran consumo de energía. Por ejemplo, un nuevo supercomputador llamado Aquasar tendrá una velocidad tope de 10 teraflops. Mientras tanto el consumo de energía de un solo rack de este supercomputador consume cerca de 10 kW. Como comparación, un rack del supercomputador Blue Gene L/P consume alrededor de 40 kW.

El consumo promedio de un supercomputador dentro de la lista de los 500 supercomputadores más rápidos del mundo es de alrededor de 257 kW.

Para el supercomputador Aquasar, que será instalado en el Instituto Tecnológico Federal Suizo (ETH), se utilizará un nuevo diseño de enfriamiento líquido. Se necesitarán 10 litros de agua que fluirán a una tasa de 29,5 litros por minuto.

En el caso del ETH en Suiza, el calor extraído del supercomputador será reciclado para calentar habitaciones dentro de la misma universidad.

En 2019 el segundo supercomputador (American Sierra) en la lista TOP500 consumía la mitad de energía que el tercero en la lista (Sunway TaihuLight).¹

Características

Las principales son:

- Velocidad de procesamiento: miles de millones de instrucciones de coma flotante por segundo.
- Usuarios a la vez: hasta miles, en entorno de redes amplias.
- Tamaño: requieren instalaciones especiales y aire acondicionado industrial.
- Dificultad de uso: solo para especialistas.
- Clientes usuales: grandes centros de investigación.
- Penetración social: prácticamente nula.
- Impacto social: muy importante en el ámbito de la investigación, ya que provee cálculos a alta velocidad de procesamiento, permitiendo, por ejemplo, calcular en secuencia el genoma humano, número π , desarrollar cálculos de problemas físicos dejando un margen de error muy bajo, etc.
- Parques instalados: menos de un millar en todo el mundo.
- Hardware: Principal funcionamiento operativo

Principales usos

Las supercomputadoras se utilizan para abordar problemas muy complejos o que no pueden realizarse en el mundo físico bien, ya sea porque son peligrosos, involucran cosas increíblemente pequeñas o increíblemente grandes. A continuación, damos algunos ejemplos:

- Mediante el uso de supercomputadoras, los investigadores modelan el clima pasado y el clima actual y predicen el clima futuro.
- Los científicos que investigan el espacio exterior y sus propiedades utilizan las supercomputadoras para simular los interiores estelares, simular la evolución estelar de las estrellas (eventos de supernova, colapso de nubes moleculares, etc.), realizar simulaciones cosmológicas y modelar el clima espacial.
- Los científicos usan supercomputadoras para simular de qué manera un tsunami podría afectar una determinada costa o ciudad.
- Las supercomputadoras se utilizan para probar la aerodinámica de los más recientes aviones militares.
- Las supercomputadoras se están utilizando para modelar cómo se doblan las proteínas y cómo ese plegamiento puede afectar a la gente que sufre la enfermedad de Alzheimer, la fibrosis quística y muchos tipos de cáncer.
- Las supercomputadoras se utilizan para modelar explosiones nucleares, limitando la necesidad de verdaderas pruebas nucleares.

2) Golpe cibernético global - Alfredo Jalife Rahme

El derribo hollywoodense el 11 de septiembre de 2 torres gemelas —luego la de un tercer inmueble por la tarde—, que al parecer terminó "implosión controlada", desembocó en el principio de la restricción de las libertades primordiales en EEUU con el republicano Baby Bush, además de sus 2 guerras fallidas en Irak y Afganistán para el control del petróleo de Oriente Medio.

Hace 9 años, Leon Panetta, secretario de Protección con Obama, advirtió que EEUU vivía un "instante cibernético pre-Pearl Harbor 11/9, una vez que 4 actores —Rusia, China, Irán e identidades terroristas sin especificar— se disponen a golpear la crítica infraestructura de EEUU".

Washington intentó obligar su modelo SOPA (Stop En línea Piracy Act) que causó la revolución de la ciudadanía, por lo cual Panetta, el exsecretario de Custodia, impulsó CISP (Cyber Intelligence Sharing and Protection): "iniciativa de ley por orden ejecutiva sin anuencia del Congreso y de los habitantes, lo que posibilita compartir el tráfico de información de internet entre el Regimen de EEUU y privilegiadas transnacionales tecnológicas.

De esta forma la grotesca toma del Capitolio está siendo explotada por los pletóricos enemigos de Trump del complejo Pentágono - Deep State - Wall Street - Silicon Valley, incluyendo al eje demócrata de los Clinton/los Obama aliado a George Soros y a un nada desdeñable conjunto de poderosos republicanos: Baby Bush, Mitt Romney, Liz Cheney (hija del exvicepresidente Dick Cheney), etcétera.

Jatras avizora que "desde el 20 de enero, seguirá una ráfaga de ocupaciones ejecutivas y legislaciones para quitar los últimos vestigios de lo cual ha sido un territorio independiente" una vez que "la Primera Enmienda (libertad de expresión, de creencia y de asociación) son solo formalidades ahora y la Segunda Enmienda (derecho a portar armas, lo que es considerado importante al criterio de Estados Unidos de ciudadanía libre) está en serio riesgo".

El Pentágono - Deep State - Wall Street - Silicon Valley conocidos como los pletóricos enemigos de Trump, se les atribuye a ellos la acción de censura hacia Donald Trump, por medio del accionar del Big Tech del GAFAM (Google/ Apple/ Facebook/ Amazon/ Microsoft) Twitter que ha emergido como una omnipotente ciberocracia que ha propinado un golpe cibernético a nivel mundial y que controlará a la mayoría de los países valetudinarios que fueron hechos prisioneros en sus redes.

Con esto podemos decir que Donald Trump y todos sus seguidores fueron censurados por esta organización o ciberocracia llamada GAFAM, la cual se creó con

el fin de atacar a el llamado movimiento “pre-Pearl Harbor 11/9” quien contaba con cuatro actores: Rusia, China, Irán e identidades terroristas sin especificar.

3) Test de personalidad

Hacer el test de personalidad, después leer a detalle la descripción completa de la personalidad y redactar su opinión del resultado

Personalidad “Cónsul”

“Anímense, elévense y fortalézcanse unos a otros. Porque la energía positiva irradiada hacia una persona será percibida por todos nosotros.”

DEBORAH DAY

Las personas que comparten la personalidad de Cónsul son, ya que no hay palabra mejor, populares, lo cual tiene sentido, ya que también se trata de un tipo de personalidad muy común, que representa el doce por ciento de la población. En la escuela secundaria, las Cónsules son las animadores y los quarterback del equipo, los que establecen la pauta, los que son el centro de atención y dirigen a sus equipos hacia el triunfo y la fama. Con la edad, a los Cónsules les sigue gustando apoyar a sus amigos y seres queridos, organizar reuniones sociales y hacer todo lo

Discutir sobre teorías científicas o debatir la política europea no captará el interés de un Cónsul durante mucho tiempo. A los individuos con la personalidad de Cónsul les importan más los asuntos tangibles y prácticos, además de mejorar su status social y observar a los demás. Estar al tanto de todo lo que pasa a su alrededor es su plato preferido, pero las personalidades tipo Cónsul hacen todo lo posible para utilizar sus poderes para el bien.

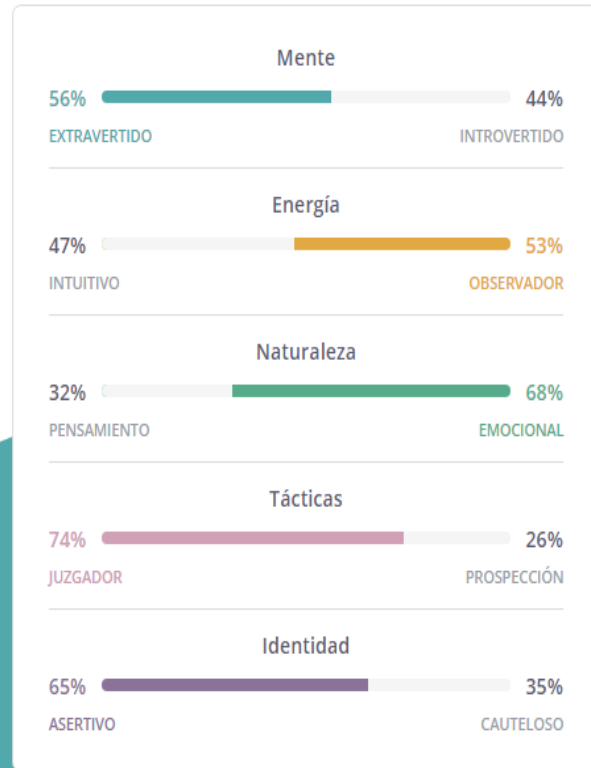
Respetar la sabiduría del liderazgo

Los Cónsules son altruistas y se toman en serio su responsabilidad de ayudar y hacer lo correcto. A diferencia de otros tipos de personalidad más idealistas, las personas con personalidad de Cónsul basarán su brújula moral en las tradiciones establecidas y en las leyes, en respetar la autoridad y las reglas en lugar de apoyar su moralidad en la filosofía o el misticismo. Sin embargo, es importante para la personalidad de Cónsul recordar que las personas provienen de diversos orígenes y perspectivas y que lo que puede parecer correcto para ellos no siempre es la verdad absoluta.

Tu tipo de personalidad es:

Cónsul

ESFJ-A



Desde mi punto de vista, estos tipos de “test” no ayudan mucho a descubrir la personalidad de las personas al 100% puesto que solo son paginas programables en base a la ideología psicológica de las personas, pero no a la ideología de la personalidad certera. En la descripción dice que soy una persona popular y me gusta estar rodeado de amistades. Realmente esa descripción es falsa puesto que yo soy una persona muy seria, no me gusta estar acompañado de muchas personas. Soy una persona que prefiere la soledad y la tranquilidad, me gusta mucho trabajar por las noches ya que no me gusta escuchar demasiado ruido.

4) Collage digital de sus gustos y pasiones

Hacer un collage digital de sus gustos y pasiones incluyendo un párrafo de semblanza personal de 5 líneas (si trabajan, indicar su horario y actividad laboral).



Me gusta mucho la música electrónica, hacer música, practicar el arte marcial tae kwon do, nadar los 4 estilos de natación, bailar, ir a conciertos, comer toda la variedad de comida mexicana, trabajar durante la noche, saber un poco de astronomía, cocinar, me gusta trabajar, ganar mi propio dinero para poder comprarme mis propias cosas, leer, dormir y estar con mi familia. Trabajo como asesor telefónico de la compañía telefónica Movistar de Lunes a Domingo con un día de descanso rolando entre semana de 16:00 a 23:00

B) Actividades semana marzo 16-19, 2021

Actividades individuales:

1) Película The Imitation Game

The Imitation Game es una película de drama histórico estadounidense de 2014 dirigida por Morten Tyldum y escrita por Graham Moore, basada en la biografía de 1983 Alan Turing: The Enigma de Andrew Hodges. El título de la película cita el nombre del juego que el criptoanalista Alan Turing propuso para responder a la pregunta "¿Pueden pensar las máquinas?", En su artículo seminal de 1950 "Computing Machinery and Intelligence". La película está protagonizada por Benedict Cumberbatch como Turing, quien descifró los mensajes de inteligencia alemanes para el gobierno británico durante la Segunda Guerra Mundial. Keira Knightley, Matthew Goode, Rory Kinnear, Charles Dance y Mark Strong aparecen en papeles secundarios.

El juego de imitación se estrenó en cines en los Estados Unidos el 28 de noviembre de 2014. La película recaudó más de \$ 233 millones en todo el mundo con un presupuesto de producción de \$ 14 millones, lo que la convirtió en la película independiente más taquillera de 2014. Recibió ocho nominaciones en los 87 Premios de la Academia., ganadora de Mejor Guión Adaptado, cinco nominaciones en la 72ª edición de los Globos de Oro y tres nominaciones en la 21ª edición de los Screen Actors Guild Awards. También recibió nueve nominaciones al BAFTA y ganó el premio People's Choice Award en el 39º Festival Internacional de Cine de Toronto.

La película fue criticada por algunos por su descripción inexacta de los eventos históricos y por restar importancia a la homosexualidad de Turing, un elemento clave de sus luchas originales. Sin embargo, la organización de defensa de los derechos civiles LGBT Human Rights Campaign la honró por llevar el legado de Turing a una audiencia más amplia con su enfoque sutil y realista.

En Rotten Tomatoes, la película tiene una calificación de aprobación del 89% según 284 reseñas, con una calificación promedio de 7.7 / 10. El consenso crítico del sitio dice: "Con una destacada actuación protagonizada por Benedict Cumberbatch que ilumina su historia basada en hechos, The Imitation Game sirve como una entrada eminentemente bien hecha en el género 'biopic de prestigio'". En Metacritic, la película tiene una puntuación media ponderada de 73 sobre 100, basada en 49 críticas, que indica "opiniones en general favorables". La película recibió una calificación promedio poco común de "A +" de la firma de investigación de mercado CinemaScore, y una calificación de "recomendación definitiva" del 90% de su audiencia principal, según PostTrak. También se incluyó en las "10 mejores películas de 2014" del National Board of Review y del American Film Institute.

Las actuaciones de Benedict Cumberbatch y Keira Knightley obtuvieron elogios de la crítica y ambos recibieron nominaciones al Premio de la Academia como Mejor Actor y Mejor Actriz de Reparto, respectivamente.

Rex Reed, del New York Observer, declaró que "una de las historias más importantes del siglo pasado es una de las mejores películas de 2014". [59] Kaleem Aftab de The Independent le dio a la película una reseña de cinco estrellas, y la calificó como la "Mejor película británica del año". Empire lo describió como un "excelente thriller" y Glamour lo declaró "un clásico instantáneo". Peter Debruge de Variety agregó que la película está "bellamente escrita, elegantemente montada y conmovedora". El crítico Scott Foundas declaró que "la película es innegablemente fuerte en el sentido de que una luz brillante se apaga demasiado pronto, y el destino a menudo indigno de aquellos que se atreven a irritarse con las normas establecidas de la sociedad". El crítico Leonard Maltin afirmó que la película tiene "un reparto ideal con todos los roles llenos a la perfección". Se elogió la interpretación de apoyo de Knightley como Clarke, la edición de Goldenberg, la banda sonora de Desplat, la cinematografía de Faura y el diseño de producción de Djurkovic. La película fue recibida con entusiasmo en el Festival de Cine de Telluride y ganó el "Premio People's Choice a la Mejor Película" en TIFF, el premio más alto del festival.

Actividades en equipo:

2) Ensayo / Resumen Untangling the Tale of Ada Lovelace de Stephen Wolfram

Ada Lovelace nació hoy hace 200 años. Para algunos, es una gran heroína en la historia de la informática; para otros una figura menor sobreestimada. He tenido curiosidad durante mucho tiempo sobre cuál es la verdadera historia. Y en preparación para su bicentenario, decidí intentar resolver lo que para mí siempre ha sido el "misterio de Ada".

Fue mucho más difícil de lo que esperaba. Los historiadores no están de acuerdo. Las personalidades de la historia son difíciles de leer. La tecnología es difícil de entender. Toda la historia está entrelazada con las costumbres de la alta sociedad británica del siglo XIX. Y hay una sorprendente cantidad de desinformación e interpretación errónea por ahí.

Pero después de un poco de investigación, incluida la de ver muchos documentos originales, siento que finalmente he llegado a conocer a Ada Lovelace y he comprendido su historia. En cierto modo, es una historia ennobecedora e inspiradora; de alguna manera es frustrante y trágico.

Es una historia compleja, y para entenderla, tendremos que comenzar repasando muchos hechos y narrativas.

La vida temprana de Ada

Empecemos por el principio. Ada Byron, como la llamaban entonces, nació en Londres el 10 de diciembre de 1815 de padres de la alta sociedad recién casados. Su padre, Lord Byron (George Gordon Byron) tenía 27 años y acababa de alcanzar el estatus de estrella de rock en Inglaterra por su poesía. Su madre, Annabella Milbanke, era una heredera de 23 años comprometida con causas progresistas, que heredó el título de baronesa Wentworth. Su padre dijo que le dio el nombre de "Ada" porque "es corto, antiguo, vocálico".

Los padres de Ada eran una especie de estudio de los opuestos. Byron tuvo una vida salvaje, y quizás se convirtió en el mejor "chico malo" del siglo XIX, con episodios oscuros en la infancia y muchos excesos románticos y de otro tipo. Además de escribir poesía y burlarse de las normas sociales de su tiempo, a menudo hacía lo inusual: tener un oso domesticado en sus habitaciones de la universidad en Cambridge, vivir con poetas en Italia y "cinco pavos reales en la gran escalera", escribir un libro de gramática en armenio y, si no hubiera muerto demasiado pronto, liderando tropas en la guerra de independencia griega (como se celebra con una gran estatua en Atenas), a pesar de no tener ningún entrenamiento militar.

Annabella Milbanke era una mujer culta, religiosa y bastante adecuada, interesada en la reforma y las buenas obras, y Byron la apodó "Princesa de los Paralelogramas". Su breve matrimonio con Byron se vino abajo cuando Ada tenía solo 5 semanas de edad, y Ada nunca volvió a ver a Byron (aunque él tenía una foto de ella en su escritorio y la mencionaba en su poesía). Murió a la edad de 36 años, en el apogeo de su fama, cuando Ada tenía 8. Había suficiente escándalo a su alrededor para alimentar cientos de libros, y la batalla de relaciones públicas entre los partidarios de Lady Byron (como la madre de Ada se llamaba a sí misma) y de él duró un siglo o más.

Ada tuvo una infancia aislada en las fincas de campo alquiladas por su madre, con institutrices y tutores y su gato mascota, la Sra. Puff. Su madre, a menudo ausente por varias curas de salud (bastante locas), impuso un sistema de educación para Ada que implicaba largas horas de estudio y ejercicios de autocontrol. Ada aprendió historia, literatura, idiomas, geografía, música, química, costura, taquigrafía y matemáticas (enseñadas en parte a través de métodos experimentales) hasta el nivel de geometría elemental y álgebra. Cuando Ada tenía 11 años, fue con su madre y un séquito en una gira de un año por Europa. Cuando regresó, estaba haciendo cosas con entusiasmo como estudiar lo que ella llamaba "flyología", e imaginar cómo imitar el vuelo de las aves con máquinas de vapor.

Pero luego se enfermó de sarampión (y quizás encefalitis) y terminó postrada en cama y con mala salud durante 3 años. Finalmente se recuperó a tiempo para seguir la costumbre de las chicas de sociedad de la época: a los 17 años se fue a Londres para una temporada de socialización. El 5 de junio de 1833, 26 días después de que fue "presentada en la corte" (es decir, conoció al rey), fue a una fiesta en la casa de Charles Babbage, de 41 años (cuyo hijo mayor tenía la misma edad que Ada). Aparentemente, ella encantó al anfitrión, y él la invitó a ella y a su madre a regresar para una demostración de su Motor Diferencial recién construido: un artilugio de 2 pies de altura con manivela con 2000 piezas de latón, que ahora se puede ver en el Museo de Ciencias en Londres:

C) Actividades semana abril 12-16, 2021

Actividades individuales:

1) Artículo Translator (computing) de Wikipedia.

Un traductor o procesador de lenguaje de programación es un término genérico que puede referirse a cualquier cosa que convierta código de un lenguaje de computadora a otro. Un programa escrito en un lenguaje de alto nivel se llama programa fuente. Estos incluyen traducciones entre lenguajes de computadora de alto nivel y legibles por humanos, como C ++ y Java, lenguajes de nivel intermedio como el código de bytes de Java, lenguajes de bajo nivel como el lenguaje ensamblador y el código de máquina, y entre niveles similares de lenguaje en diferentes sistemas informáticos. plataformas, así como de cualquiera de los anteriores a otro.

El término también se usa para traductores entre implementaciones de software e implementaciones de hardware (microchips ASIC) del mismo programa, y de descripciones de software de un microchip a las puertas lógicas necesarias para construirlo.

Diferentes tipos de traductores

Hay 3 tipos diferentes de traductores de la siguiente manera:

Compilador

Artículo principal: compilador

Un compilador es un traductor que se utiliza para convertir un lenguaje de programación de alto nivel en un lenguaje de programación de bajo nivel. Convierte todo el programa en una sesión e informa de los errores detectados después de la conversión. El compilador necesita tiempo para hacer su trabajo, ya que traduce el código de alto nivel al código de nivel inferior de una vez y luego lo guarda en la memoria. Un compilador depende del procesador y de la plataforma. Se ha abordado con nombres alternativos como los siguientes: compilador especial, compilador cruzado y compilador de fuente a fuente.

Interprete

Artículo principal: Intérprete (informática)

El intérprete es similar a un compilador, ya que es un traductor que se utiliza para convertir un lenguaje de programación de alto nivel en un lenguaje de programación de bajo nivel. La diferencia es que convierte el programa una línea de código a la vez y reporta errores cuando los detecta, mientras también realiza la conversión. Un intérprete es más rápido que un compilador, ya que ejecuta el código

inmediatamente después de leerlo. A menudo se utiliza como herramienta de depuración para el desarrollo de software, ya que puede ejecutar una sola línea de código a la vez. Un intérprete también es más portátil que un compilador, ya que es independiente del procesador, puede trabajar entre diferentes arquitecturas de hardware.

Ensamblador

Artículo principal: lenguaje ensamblador § ensamblador

Un ensamblador es un traductor que se utiliza para traducir el lenguaje ensamblador al lenguaje de máquina. Tiene la misma función que un compilador para el lenguaje ensamblado, pero funciona como un intérprete. El lenguaje ensamblador es difícil de entender ya que es un lenguaje de programación de bajo nivel. Un ensamblador traduce un lenguaje de bajo nivel, como un lenguaje ensamblador, a un lenguaje de nivel aún más bajo, como el código máquina.

2) Artículo The Latin American Supercomputing Ecosystem for Science de la revista Communications de la ACM.

big trends

DOI:10.1145/3419977

BY ISIDORO GITLER, ANTÔNIO TADEU A. GOMES,
AND SERGIO NESMACHNOW

The Latin American Supercomputing Ecosystem for Science

LARGE, EXPENSIVE, COMPUTING-INTENSIVE research initiatives have historically promoted high-performance computing (HPC) in the wealthiest countries, most notably in the U.S., Europe, Japan, and China. The exponential impact of the Internet and of artificial intelligence (AI) has pushed HPC to a new level, affecting economies and societies worldwide. In Latin America, this was no different. Nevertheless, the use of HPC in science affected the countries in the region in a heterogeneous way. Since the first edition in 1993 of the TOP500 list of most powerful supercomputing systems in the world, only Mexico and Brazil (with 18 appearances each) made the list with research-oriented supercomputers. As of June 2020, Brazil was the only representative of Latin America on the list.

HPC represents a strategic resource for Latin American researchers to respond to the economical and societal challenges in the region and to cross-fertilize with researchers in the rest of the world. Nevertheless, the Latin American countries still lag behind other countries in terms of size and regularity of investments in HPC. The table here compares the HPC capacity of the BRICS countries, which together represent almost half of the world population. As a reference, in 2018, South Africa's GDP was 29.1% lower than Argentina's and only 11.2% higher than Colombia's, the two countries in Latin America with largest GDPs after Brazil and Mexico. In spite of the overall picture described here, the landscape of the Latin American HPC ecosystem for science is promising, with many initiatives and outstanding concrete results.

HPC in Latin America: The Cases of Brazil, Mexico, and Uruguay
Comparing the situation of HPC in three different countries in Latin America helps understanding the region's distinctions, not only in terms of overall capacity, but also in terms of adopted policies for creating and operating this kind of scientific instrumentation systems. The presented examples are representative of other important initiatives in the region, for example, NLHPC in Chile, Tupac in Argentina, SC3UIS in Colombia, and CeNAT in Costa Rica.

In Brazil, the National Laboratory for Scientific Computing (LNCC) is the major player for HPC services to the scientific community. LNCC is a public, interdisciplinary research center with a mission-oriented approach to computational and mathematical modeling and simulation of complex problems. LNCC coordinates a network of 10 HPC centers (SINAPAD) funded by the Brazilian Ministry of Science, Technology and Innovations (MCTI) (see Figure 1). The SINAPAD centers offer resources, training, and scientific portals^{8,11} to the Brazilian community.

66 COMMUNICATIONS OF THE ACM | NOVEMBER 2020 | VOL. 63 | NO. 11

Actividades en equipo:

3) Resumen de ambos artículos y grabar un video

Resumen del equipo

Cuananemi Cuanalo Mario Alberto

EL ECOSISTEMA LATINOAMERICANO DE SUPERCOMPUTACIÓN PARA LA CIENCIA

Es fundamental poder utilizar esta herramienta a la que yo puedo considerar HPC ya que para el desarrollo de nuevas tecnologías aplicadas incluso para la inteligencia artificial y así poder llegar a un desarrollo humano y estas puedan facilitar más no inutilizar al ser humano, ya que ciertas máquinas pueden llegar a reemplazar al humano en tareas complicada como por ejemplo a los doctores que operan y se tenga un margen de error mínimo y se puedan salvar vidas así como llegar a tratados con los diferentes gobiernos, que lo veo muy complicado por la carrera tecnológica que se está viviendo en estos tiempos, puesto de esta ,manera podemos deducir y decir que todo desarrollo viene con implicaciones para el ser humano pero valorando los beneficios que se obtienen es la mejor forma de desarrollar la Computación de alto nivel, hay varios países

TRADUCTOR (INFORMÁTICA)

Es un intermediario entre el lenguaje fuente y el lenguaje y el lenguaje objeto, que básicamente tomado de las palabras de ciertos foros Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, es decir programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (lenguaje máquina). Generando un programa equivalente a capaz de interpretar. Pero todo depende del procesador.

Fermin Cruz Erik

EL ECOSISTEMA LATINOAMERICANO DE SUPERCOMPUTACIÓN PARA LA CIENCIA

HPC en América Latina: los casos de Brasil, México y Uruguay

Comparar la situación de HPC en tres países diferentes de América Latina ayuda a comprender las diferencias de la región, no solo en términos de capacidad general, sino también en términos de políticas adoptadas para crear y operar este tipo de sistemas de instrumentación científica. Los ejemplos presentados son representativos de otras

iniciativas importantes en la región, por ejemplo, NLHPC en Chile, Tupac en Argentina, SC3UIS en Colombia y CeNAT en Costa Rica.

CINVESTAV es una institución pública clasificada dentro de los principales Centros Nacionales de Investigación e Instituciones de Educación de Postgrado de México y, a través de ABACUS y LANCAD, un proveedor principal de recursos de HPC para las comunidades científicas y tecnológicas en México.

CINVESTAV tiene un historial sobresaliente en cuanto a iniciativas para fomentar la interacción entre la academia, el gobierno, la industria y la sociedad, junto con una trayectoria muy exitosa de colaboración mundial. ABACUS alberga una de las principales supercomputadoras de investigación de América Latina, ubicada en el puesto 255 en la lista TOP500 de julio de 2015, con un rendimiento total actualizado de ~ 0,5 petaflops y una capacidad de almacenamiento de 1 petabyte.

TRADUCTOR (INFORMÁTICA)

Este término se usa para traductores entre implementaciones de software e implementaciones de hardware (microchips ASIC) del mismo programa, y de descripciones de software de un microchip a las puertas lógicas necesarias para construirlo.

Es un término genérico que puede referirse a cualquier cosa que convierta código de un lenguaje de computadora a otro.

Un programa escrito en un lenguaje de alto nivel se llama programa fuente. Estos incluyen traducciones entre lenguajes de computadora de alto nivel y legibles por humanos, como C ++ y Java, lenguajes de nivel intermedio como el código de bytes de Java, lenguajes de bajo nivel como el lenguaje ensamblador y el código de máquina, y entre niveles similares de lenguaje en diferentes sistemas informáticos. plataformas, así como de cualquiera de las anteriores.

Lenguajes de programación más usados del 2021:

JavaScript, Python, Java, TypeScript, C#, PHP, C++, C, Shell, Ruby, Go, Swift, Lenguaje de programación R, Visual Basic, Kotlin.



Bibliografía (top 15 lenguajes de programación):

Anónimo (2021). Los 15 lenguajes de programación más usados en 2021

Recuperado de: <https://www.crehana.com/mx/blog/web/lenguajes-de-programacion-mas-usados/>

El 18 de abril de 2021

Gutierrez Arellano Rafael

EL ECOSISTEMA LATINOAMERICANO DE SUPERCOMPUTACIÓN PARA LA CIENCIA

Tratamos de entender la forma en la que se comparan los países latinoamericanos con respecto a la tecnología y desarrollo en base a la región, gracias a esto podemos percatarnos cuales son nuestras vías de desarrollo y oportunidades que podemos tener en nuestro país, en el texto el Laboratorio de matemática implicada y DPC del Centro de investigación y Estudios Avanzados (CINVESTAV), tiene un historial sobresaliente en lo que respecta a los avances y nuevas tecnologías que dependen mucho de la ubicación territorial.



Loud es una plataforma para servicios de voz humanizados e innovadores.



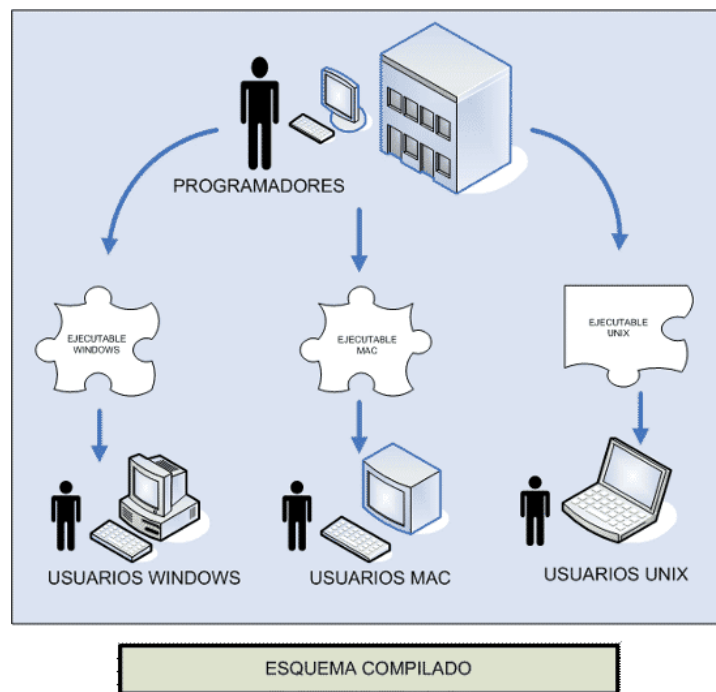
Satellogic es la primera compañía de análisis geoespacial integrada verticalmente.

PAÍS	Brazil
INDUSTRIA	Enterprise
FUNDADA EN	2019
FUNDADOR	Felipe Almeida

PAÍS	Argentina
INDUSTRIA	Spacetech
FUNDADA EN	2019
FUNDADORES	Emiliano Kargieman, Gerardo Richarte

TRADUCTOR (INFORMÁTICA)

Del traductor se desprenden 3 conceptos básicos.



Compilador, el cual transforma el código fuente de un lenguaje de programación de un lenguaje de alto nivel a lenguaje máquina, un lenguaje de bajo nivel.

Todo depende del procesador y la arquitectura de la computadora y el tiempo de trabajo del compilador.



Intérprete.

A diferencia del compilador, el intérprete no depende tanto del procesador y la arquitectura de computadora y es más portable donde la intérprete válida línea por línea el código y avisa si tienes problemas con él en alguna de las líneas de código línea por línea.

Ensamblador.

```

00001A1E 4D 4B      LDR          R3, =(stdout_ptr - 0xC000)
00001A20 E3 58      LDR          R3, [R4,R3] ; stdout
00001A22 1B 68      LDR          R3, [R3]
00001A24 18 46      MOV          R0, R3 ; stream
00001A26 FF F7 52 EA BLX          fileno
00001A2A 03 46      MOV          R3, R0
00001A2C 18 46      MOV          R0, R3
00001A2E 4A 4B      LDR          R3, =(a1ChangeDisplay - 0x1
00001A30 7B 44      ADD          R3, PC ; "1 ) Change displ
00001A32 19 46      MOV          R1, R3
00001A34 00 F0 34 FB BL          print
00001A38 48 4B      LDR          R3, =(stdin_ptr - 0xC000)
00001A3A E3 58      LDR          R3, [R4,R3] ; stdin
00001A3C 1B 68      LDR          R3, [R3]
00001A3E 18 46      MOV          R0, R3 ; stream
00001A40 FF F7 44 EA BLX          fileno
00001A44 02 46      MOV          R2, R0
00001A46 07 F5 43 63 ADD.W      R3, R7, #0xC30
00001A4A 10 46      MOV          R0, R2
00001A4C 19 46      MOV          R1, R3
00001A4E 4F F0 02 02 MOV.W      R2, #2
00001A52 4F F0 0A 03 MOV.W      R3, #0xA
00001A56 00 F0 77 FB BL          read
00001A5A 03 46      MOV          R3, R0
00001A5C 00 2B      CMP          R3, #0

```

El ensamblador se encarga de mandar con un lenguaje de bajo nivel indicaciones a la computadora y de la cual se escribe código para un funcionamiento claro con la máquina, es difícil de entender, pero muy potente a la hora de estar trabajando con él.

Resumen personal

PEREZ ARMAS FAUSTO ISAAC

Procesador de lenguaje de programación



Un traductor o procesador de lenguaje de programación es un término genérico que puede referirse a cualquier cosa que convierta código de un lenguaje de computadora a otro. Un programa escrito en un lenguaje de alto nivel se llama programa fuente. Estos incluyen traducciones entre lenguajes de computadora de alto nivel y legibles por humanos, como C ++

y Java. El término también se usa para traductores entre implementaciones de software e implementaciones de hardware (microchips ASIC) del mismo programa, y de descripciones de software de un microchip a las puertas lógicas necesarias para construirlo.

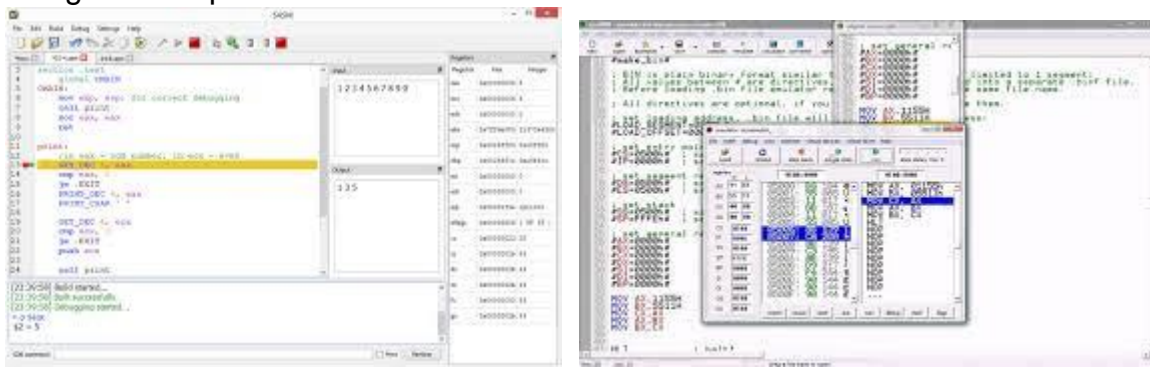
Un compilador es un traductor que se utiliza para convertir un lenguaje de programación de alto nivel en un lenguaje de programación de bajo nivel de manera general y en una sola ocasión. Convierte todo el programa en una sesión e informa de los errores detectados después de la conversión. El compilador necesita tiempo para hacer su trabajo, ya que traduce el código de alto nivel al código de nivel inferior de una vez y luego lo guarda en la memoria. Un compilador depende del procesador y de la plataforma. Se ha abordado con nombres alternativos como los siguientes: compilador especial, compilador cruzado y compilador de fuente a fuente.



El intérprete es similar a un compilador, ya que es un traductor que se utiliza para convertir un lenguaje de programación de alto nivel en un lenguaje de programación de bajo nivel. La diferencia es que convierte el programa una línea de código a la vez y reporta errores cuando los detecta, mientras también realiza la conversión. Un intérprete es más rápido que un compilador, ya que ejecuta el código

inmediatamente después de leerlo. A menudo se utiliza como herramienta de depuración para el desarrollo de software, ya que puede ejecutar una sola línea de código a la vez. Un intérprete también es más portátil que un compilador, ya que es independiente del procesador, puede trabajar entre diferentes arquitecturas de hardware.

Un ensamblador es un traductor que se utiliza para traducir lenguaje ensamblador a lenguaje de máquina. Tiene la misma función que un compilador para el lenguaje ensamblador, pero funciona como un intérprete. Es decir, es la unión de los 2 anteriores. El lenguaje ensamblador es difícil de entender ya que es un lenguaje de programación de bajo nivel. Un ensamblador traduce un lenguaje de bajo nivel, como un lenguaje ensamblador, a un lenguaje de nivel aún más bajo, como el código de máquina.



En México, ABACUS, el Laboratorio de Matemática Aplicada y HPC del Centro de Investigación y Estudios Avanzados (CINVESTAV) ejemplifica las diversas iniciativas de HPC agrupadas en la Red Mexicana en HPC (REDMEXSU), cuyos principales miembros se muestran en la Figura 1.

El CINVESTAV es una institución pública clasificada dentro de los principales Centros Nacionales de Investigación e Instituciones de Educación de Postgrado de

México y, a través de ABACUS y LANCAD, un proveedor principal de recursos de HPC para las comunidades científicas y tecnológicas en México. CINVESTAV tiene un historial sobresaliente en cuanto a iniciativas para fomentar la interacción entre la academia, el gobierno, la industria y la sociedad, junto con una trayectoria muy exitosa de colaboración mundial. ABACUS alberga una de las principales



supercomputadoras de investigación de América Latina, ubicada en el puesto 255 en la lista TOP500 de julio de 2015

IMÁGENES DEL VIDEO





Inicio Insertar Diseño Disposición Referencias Correspondencia Revisar Vista Ayuda ¿Qué desea hacer?

COMPILADOR. INTERPRETE.

¿QUÉ ES UN COMPILADOR?

TRADUCTOR

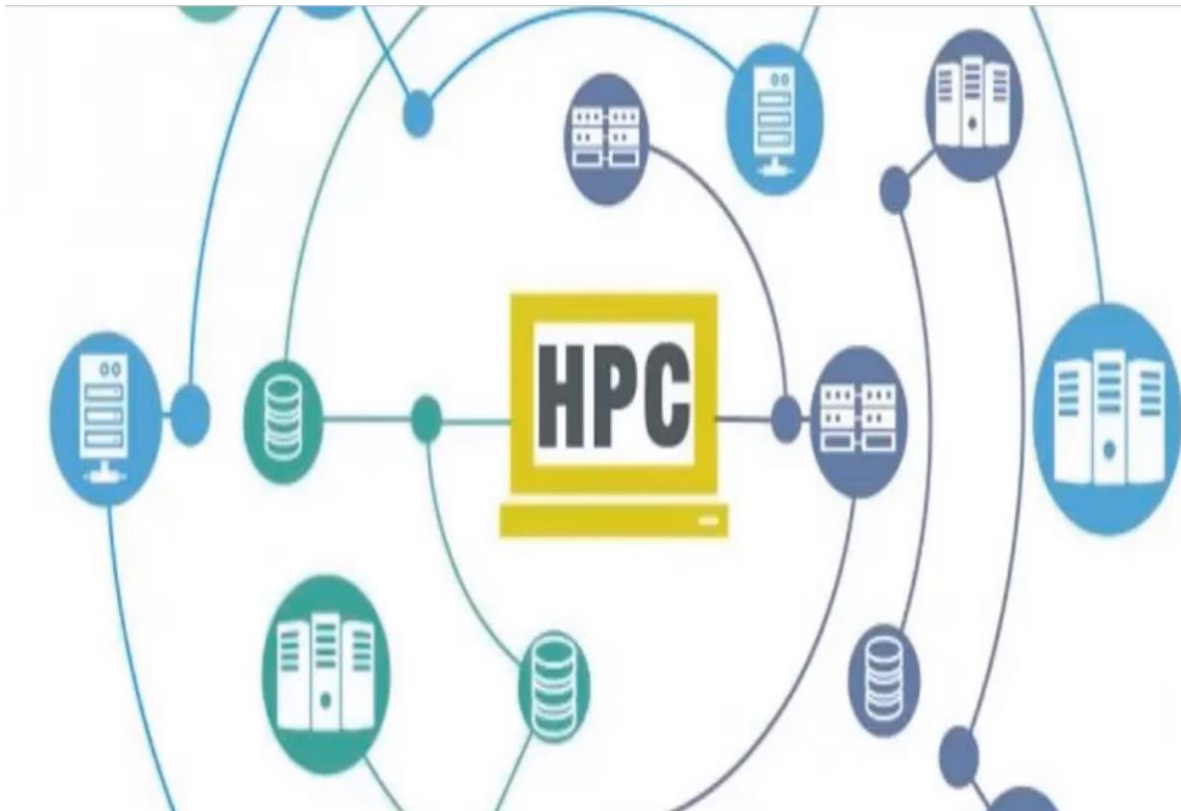
INTERPRETE

TRADUCTOR INFORMATICO.

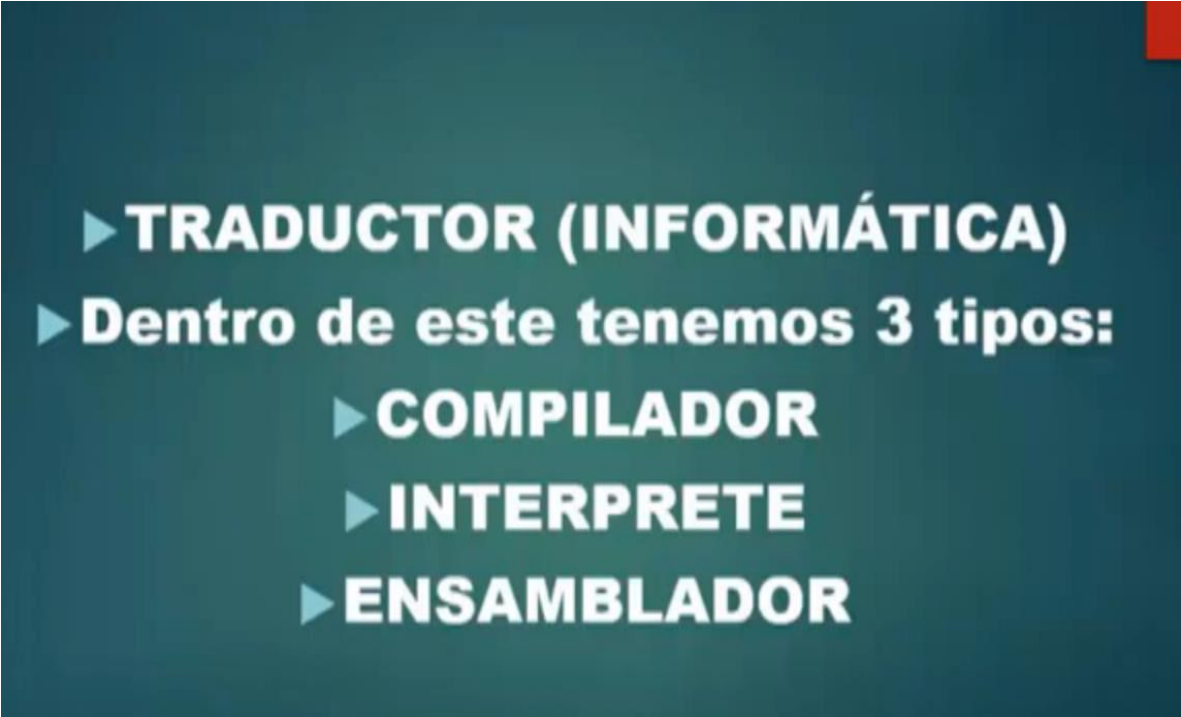
ENSAMBLADOR.

MOV	0x_0108
MOV	AX,09
INT	21
MOV	AX,00
INT	21

Hola, este es un programa hecho en ensamblador para la Wikipedia



► EL ECOSISTEMA LATINOAMERICANO DE SUPERCOMPUTACIÓN PARA LA CIENCIA

- 
- ▶ **TRADUCTOR (INFORMÁTICA)**
 - ▶ **Dentro de este tenemos 3 tipos:**
 - ▶ **COMPILADOR**
 - ▶ **INTERPRETE**
 - ▶ **ENSAMBLADOR**

LIGA DEL VIDEO:

<https://youtu.be/qJyOnUaaDac>

D) Actividades semana abril 19-23, 2021

Actividades individuales

Introduction" y "Compiler Structure". Capítulo 1 del libro "Engineering a Compiler" de Cooper y Torczon (2012).

INTRODUCCIÓN

El papel de la computadora en la vida diaria crece cada año. Con el surgimiento de la Internet, las computadoras y el software que se ejecuta en ellas brindan comunicaciones, noticias, entretenimiento y seguridad. Las computadoras integradas han cambiado las formas en que construimos automóviles, aviones, teléfonos, televisores y radios. La computación ha creado categorías de actividad completamente nuevas, desde videojuegos a redes sociales. Las supercomputadoras predicen el clima diario y el curso de violentas tormentas. Las computadoras integradas sincronizan los semáforos y entregan el correo electrónico a su bolsillo. Todas estas aplicaciones informáticas se basan en programas informáticos de software. que construyen herramientas virtuales sobre las abstracciones de bajo nivel proporcionadas por el hardware subyacente. Casi todo ese software es traducido por una herramienta llamado compilador. Un compilador es simplemente un programa de computadora que trans- Compilador un programa de computadora que traduce otros programas de computador late otros programas informáticos para prepararlos para su ejecución.

Hoja de ruta conceptual

Un compilador es una herramienta que traduce software escrito en un idioma a otro idioma. Para traducir texto de un idioma a otro, la herramienta debe comprender tanto la forma o la sintaxis como el contenido o el significado del idioma de entrada. Necesita comprender las reglas que gobiernan la sintaxis y el significado en el lenguaje de salida. Finalmente, necesita un esquema para mapear contenido. del idioma de origen al idioma de destino. La estructura de un compilador típico se deriva de estas simples observaciones. El compilador tiene una interfaz para manejar el lenguaje fuente. Tiene espalda terminar para tratar con el idioma de destino. Conectando la parte delantera y trasera Al final, tiene una estructura formal para representar el programa en una forma intermedia cuyo significado es en gran medida independiente de cualquiera de los dos idiomas. A mejorar la traducción, un compilador a menudo incluye un optimizador que analiza y reescribe esa forma intermedia.

Descripción general

Los programas de computadora son simplemente secuencias de operaciones abstractas escritas en un lenguaje de programación: un lenguaje formal diseñado para expresar computación. Los lenguajes de programación tienen propiedades y significados rígidos, como opuesto a los lenguajes naturales, como el chino o el portugués. Programación los lenguajes están diseñados para la expresividad, la concisión y la claridad. Natural los idiomas permiten la ambigüedad. Los lenguajes de programación están diseñados para evitar ambigüedad; un programa ambiguo no tiene sentido. Lenguajes de programación están diseñados para especificar cálculos, para registrar la secuencia de acciones que realizar alguna tarea o producir algunos resultados.

Los lenguajes de programación están, en general, diseñados para permitir que los humanos expresen cálculos como secuencias de operaciones. Procesadores de computadora, en adelante De nominados procesadores, microprocesadores o máquinas, están diseñados para ejecutar secuencias de operaciones. Las operaciones que implementa un procesador tienen, en su mayor parte, un nivel de abstracción mucho más bajo que los especificados en un lenguaje de programación. Por ejemplo, un lenguaje de programación normalmente incluye una forma concisa de imprimir un número en un archivo. Ese soltero La declaración del lenguaje de programación debe traducirse literalmente a cientos de las operaciones de la máquina antes de que pueda ejecutarse.

La herramienta que realiza tales traducciones se llama compilador. El compilador toma como entrada un programa escrito en algún lenguaje y produce como salida un programa equivalente. En la noción clásica de un compilador, la salida El programa se expresa en las operaciones disponibles en algún procesador específico, a menudo llamado la máquina de destino. Visto como una caja negra, un compilador podría se parece a esto:

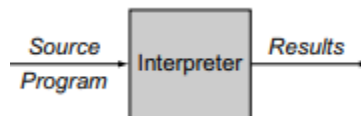


Los lenguajes "fuente" típicos pueden ser c, c ++, fortran, Java. El idioma "de destino" suele ser el conjunto de instrucciones de algún procesador. Conjunto de instrucciones El conjunto de operaciones soportadas por un procesador; el diseño general de un conjunto de instrucciones es a menudo llamada arquitectura de conjunto de instrucciones o ISA. Algunos compiladores producen un programa de destino escrito en un lenguaje de programación orientado a humanos en lugar del lenguaje ensamblador de alguna computadora. Los programas que producen estos compiladores requieren una traducción adicional antes pueden ejecutarse directamente en una computadora. Muchos compiladores de investigación producen

Programas en C como salida. Debido a que los compiladores de C están disponibles en la mayoría de computadoras, esto hace que el programa de destino sea ejecutable en todos esos sistemas, a costa de una compilación adicional para el objetivo final. Los compiladores que se dirigen a lenguajes de programación en lugar del conjunto de instrucciones de una computadora son a menudo llamados traductores fuente a fuente.

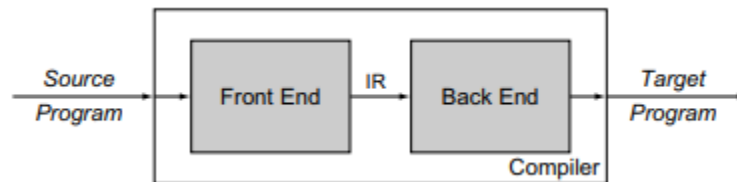
Muchos otros sistemas califican como compiladores. Por ejemplo, un programa de composición tipográfica que produce PostScript puede considerarse un compilador. Toma como Ingrese una especificación de cómo debe verse el documento en la página impresa y produce como salida un archivo PostScript. PostScript es simplemente un lenguaje para describir imágenes. Porque el programa de tipografía toma un ejecutable, especificación y produce otra especificación ejecutable, es un compilador.

El código que convierte PostScript en píxeles suele ser un intérprete, no un compilador. Un intérprete toma como entrada una especificación ejecutable y produce como salida el resultado de ejecutar la especificación.



ESTRUCTURA DEL COMPILADOR

Un compilador es un sistema de software grande y complejo. La comunidad ha sido compiladores de construcción desde 1955, y a lo largo de los años, hemos aprendido muchas lecciones sobre cómo estructurar un compilador. Anteriormente, describimos un compilador como un cuadro simple que traduce un programa fuente en un programa de destino. Realidad, por supuesto, es más complejo que esa simple imagen. Como sugiere el modelo de caja única, un compilador debe comprender el programa fuente que toma como entrada y mapea su funcionalidad al destino máquina. La naturaleza distinta de estas dos tareas sugiere una división del trabajo y conduce a un diseño que descompone la compilación en dos piezas principales: una front end y back end.



La interfaz se centra en comprender el programa en el idioma de origen. La El back-end se enfoca en mapear programas a la máquina de destino. Esta separación de preocupaciones tiene varias implicaciones importantes para el diseño e implementación de compiladores.

La interfaz debe codificar su conocimiento del programa fuente en algunos Estructura IR para uso posterior por parte del back-end. Esta representación intermedia (ir) Un compilador usa un conjunto de estructuras de datos para representan el código que procesa. Esa forma es llamada representación intermedia, o IR. se convierte en la representación definitiva del compilador del código que está traduciendo. En cada punto de la compilación, el compilador tendrá una representación definitiva. De hecho, puede usar varios IR diferentes a medida que avanza la compilación, pero en cada punto, una representación será la ir definitiva. Pensamos en el ir definitivo como la versión del programa pasada entre independientes Fases del compilador, como el ir pasó del front-end al back-end en el dibujo anterior

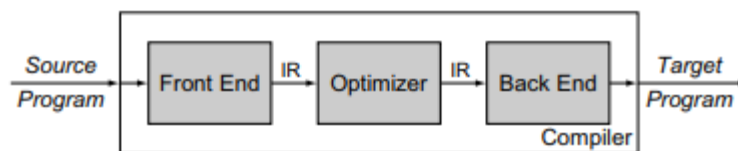
En un compilador de dos fases, la interfaz debe garantizar que el programa fuente está bien formado y debe mapear ese código en el archivo ir. El back-end debe mapear el programa ir en el conjunto de instrucciones y los recursos finitos del objetivo máquina. Debido a que el back-end solo procesa lo creado por el front-end, puede sumir que el ir no contiene errores sintácticos o semánticos. El compilador puede realizar varias pasadas sobre la forma ir del código antes emitiendo el programa de destino. Esto debería conducir a un mejor código, ya que el compilador

puede, en efecto, estudiar el código en una fase y registrar detalles relevantes. Luego, en fases posteriores, puede utilizar estos hechos registrados para mejorar la calidad de traducción. Esta estrategia requiere que el conocimiento derivado en el primer paso sea registrado en el ir, donde los pasos posteriores pueden encontrarlo y usarlo. Finalmente, la estructura de dos fases puede simplificar el proceso de retargeting la intérprete válida

La tarea de cambiar el compilador para generar el código para un nuevo procesador a menudo se llama Reorientar el compilador. el compilador. Podemos imaginar fácilmente la construcción de múltiples backends para una interfaz única para producir compiladores que acepten el mismo lenguaje, pero se dirijan a diferentes máquinas. Del mismo modo, podemos imaginar las interfaces para diferentes

Descripción general de la compilación

lenguajes que producen el mismo ir y usan un back-end común. Ambas cosas
Los escenarios suponen que un ir puede servir para varias combinaciones de fuentes. y objetivo; en la práctica, tanto los detalles específicos del lenguaje como los específicos de la máquina generalmente encuentran su camino hacia el ir. La introducción de un ir permite agregar más fases a la compilación. Leal escritor del compilador puede insertar una tercera fase entre el front-end y el back Final del optimizador. Esta sección central, u optimizador, toma un programa ir como entrada y La sección central de un compilador, llamada O optimizador, analiza y transforma su IR a mejorarlo, produce un programa ir semánticamente equivalente como salida. Usando el ir como interfaz, el escritor del compilador puede insertar esta tercera fase con un mínimo interrupción en la parte delantera y trasera. Esto conduce al siguiente compilador estructura, denominada compilador trifásico.

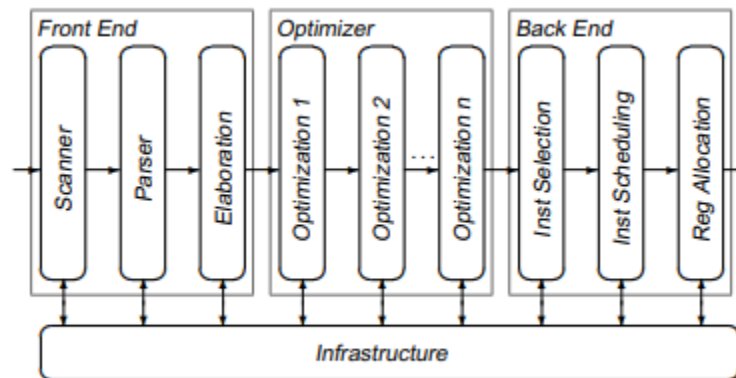


El optimizador es un transformador ir a ir que intenta mejorar el programa ir de alguna manera. (Tenga en cuenta que estos transformadores son, en sí mismos, compiladores de acuerdo con nuestra definición en la Sección 1.1.) El optimizador puede hacer uno o más pasas sobre el ir, analiza el ir y reescribe el ir. El optimizador puede reescribir el ir de una manera que probablemente produzca un programa de destino más rápido desde el back-end o un programa de destino más pequeño desde el back-end. Puede tener otros objetivos, como un programa que produce menos errores de página o usa menos energía.

Conceptualmente, la estructura trifásica representa la optimización clásica compilador. En la práctica, cada fase se divide internamente en una serie de pasos.

La parte delantera consta de dos o tres pasadas que manejan los detalles de reconocer programas válidos en el idioma de origen y producir la información inicial forma del programa. La sección central contiene pases que realizan diferentes optimizaciones. El número y el propósito de estos pases varían de compilador a compilador. El back-end consta de una serie de pases, cada uno de los cuales lo que lleva el programa ir un paso más cerca del conjunto de instrucciones de la máquina de destino. Las tres fases y sus pases individuales comparten una infraestructura. Esta estructura se muestra en la Figura 1.1.

En la práctica, la división conceptual de un compilador en tres fases, una end, una sección intermedia u optimizador, y un back-end, es útil. Los problemas abordados por estas fases son diferentes. La parte delantera se ocupa de comprender el programa fuente y registrar los resultados de su análisis en su forma. La sección del optimizador se centra en mejorar la forma de ir.



■ FIGURE 1.1 Structure of a Typical Compiler.

Actividades en equipo

2) "Getting Started / Tutorials" de la documentación oficial de LLVM.

El proyecto LLVM tiene múltiples componentes. El núcleo del proyecto en sí mismo se llama "LLVM". Contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para procesar representaciones intermedias y convertirlas en archivos de objeto. Las herramientas incluyen un ensamblador, un desensamblador, un analizador de código de bits y un optimizador de código de bits. También contiene pruebas de regresión básicas.

Hay muchos proyectos diferentes que componen LLVM. La primera pieza es la suite LLVM. Contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para usar LLVM. Contiene un ensamblador, desensamblador, analizador de código de bits y optimizador de código de bits. También contiene pruebas de regresión básicas que se pueden utilizar para probar las herramientas LLVM y la interfaz de Clang.

La segunda pieza es la parte delantera de Clang. Este componente compila código C, C ++, Objective C y Objective C ++ en código de bits LLVM. Clang generalmente usa bibliotecas LLVM para optimizar el código de bits y emitir código de máquina. LLVM es totalmente compatible con el formato de archivo de objeto COFF, que es compatible con todas las demás cadenas de herramientas de Windows existentes.

La última parte importante de LLVM, la ejecución de Test Suite, no se ejecuta en Windows, y este documento no lo analiza.

Puede encontrar información adicional sobre la estructura de directorios de LLVM y la cadena de herramientas en la página principal Introducción al sistema LLVM.

Requisitos

Antes de comenzar a utilizar el sistema LLVM, revise los requisitos que se indican a continuación. Esto puede ahorrarle algunos problemas al saber de antemano qué hardware y software necesitará.

Hardware

Cualquier sistema que pueda ejecutar adecuadamente Visual Studio 2017 está bien. El árbol de origen de LLVM y los archivos de objeto, las bibliotecas y los ejecutables consumirán aproximadamente 3 GB.

Software

Necesitará Visual Studio 2017 o superior, con la última actualización instalada.

También necesitará el sistema de compilación CMake, ya que genera los archivos del proyecto que utilizará para compilar.

Si desea ejecutar las pruebas LLVM, necesitará Python. Se sabe que la versión 3.6 y posteriores funcionan. También necesitará herramientas GnuWin32.

No instale el árbol de directorio LLVM en una ruta que contenga espacios (p. Ej.) Ya que el paso de configuración fallará. C:\Documents and Settings\...

Para simplificar el procedimiento de instalación, también puede utilizar Chocolatey como administrador de paquetes. Después de la instalación de Chocolatey, ejecute estos comandos en un shell de administración para instalar las herramientas necesarias:

```
choco install -y ninja git cmake gnuwin python3
```

```
pip3 install psutil
```

También hay un Dockerfile de Windows con toda la cadena de herramientas de compilación. Esto se puede usar para probar la compilación con una cadena de herramientas diferente a la instalación de su host o para crear servidores de compilación

.

E) Actividades semana abril 26-30, 2021

Actividades individuales:

- 1) "Overview of Translation" del capítulo 1 del libro "Engineering a Compiler" de Cooper y Torczon (2012).

RESUMEN DE LA TRADUCCIÓN

Para traducir código escrito en un lenguaje de programación en código adecuado para ejecución en alguna máquina de destino, un compilador se ejecuta a través de muchos pasos. Para que este proceso abstracto sea más concreto, considere los pasos necesarios para generar código ejecutable para la siguiente expresión:

$$a \leftarrow a \times 2 \times b \times c \times d$$

donde a, b, c y d son variables, \leftarrow indica una asignación y \times es el operador para la multiplicación.

En las siguientes subsecciones, rastreamos la ruta que toma un compilador para convertir esta simple expresión en código ejecutable.

La interfaz

Antes de que el compilador pueda traducir una expresión en código de máquina de destino ejecutable, debe comprender tanto su forma o sintaxis como su significado. La interfaz determina si el código de entrada está bien formado, en términos de sintaxis y semántica. Si encuentra que el código es válido, crea una representación del código en la representación intermedia del compilador; Si no, informa al usuario con mensajes de error de diagnóstico para identificar el problema con el código.

Comprobación de sintaxis

Para comprobar la sintaxis del programa de entrada, el compilador debe comparar la estructura del programa contra una definición del lenguaje. Esto requiere una definición formal apropiada, un mecanismo eficiente para probar si o No la entrada cumple con esa definición, y un plan sobre cómo proceder en una entrada ilegal. Matemáticamente, el idioma de origen es un conjunto, generalmente infinito, de cadenas definido por un conjunto finito de reglas, llamado gramática. Dos pases separados en el front-end, llamado escáner y analizador, determina si o no el código de entrada es, de hecho, un miembro del conjunto de programas válidos definidos por la gramática.

Las gramáticas del lenguaje de programación generalmente se refieren a palabras basadas en sus partes. del habla, a veces llamadas categorías sintácticas. Basando las reglas gramaticales en partes del discurso permite que una sola regla describa muchas oraciones. Por ejemplo, en inglés, muchas oraciones tienen la forma

Oración \rightarrow Sujeto verbo Objeto end Mark

donde el verbo y la marca final son partes del discurso, y Oración, Asunto y los objetos son variables sintácticas. La oración representa cualquier cadena con la forma descrito por esta regla. El símbolo "→" dice "deriva" y significa que una instancia de la derecha El lado de la mano se puede abstraer a la variable sintáctica en el lado izquierdo.

Considere una oración como "Los compiladores son objetos diseñados". El primer paso en la comprensión de la sintaxis de esta oración es identificar palabras distintas en el programa de entrada y para clasificar cada palabra con una parte del discurso. en un compilador, esta tarea recae en una pasada llamada escáner. El escáner lleva un escáner la pasada del compilador que convierte una cadena de personajes en un torrente de palabras Flujo de caracteres y lo convierte en un flujo de palabras clasificadas, que, es decir, pares de la forma (p, s), donde p es la parte del discurso de la palabra y s es su ortografía. Un escáner convertiría la oración de ejemplo en lo siguiente flujo de palabras clasificadas:

(sustantivo, "compiladores"), (ver o, "son"), (adjetivo, "diseñado"),
(sustantivo, "objetos"), (marca al final, ".")

En la práctica, la ortografía real de las palabras podría almacenarse en una tabla hash. y representado en pares con un índice entero para simplificar las pruebas de igualdad.

En el siguiente paso, el compilador intenta hacer coincidir el flujo de palabras categorizadas contra las reglas que especifican la sintaxis del idioma de entrada. Por ejemplo, un conocimiento práctico del inglés puede incluir los siguientes aspectos gramaticales reglas:

1 oración → Sujeto verbo Objeto endamar
2 asunto → sustantivo
3 sujeto → Modificador sustantivo
4 objeto → sustantivo
5 objeto → Modificador sustantivo
6 modificador → adjetivo
...

Mediante inspección, podemos descubrir la siguiente derivación para nuestro ejemplo oración: Oración de prototipo de regla- Oración

1 sujeto verbo Objeto endmark
2 sustantivo verbo Object endmark

5 sustantivo Verbo Modificador sustantivo endmark

6 sustantivo verbo adjetivo sustantivo endmark

La derivación comienza con la variable sintáctica Sentence. A cada paso, reescribe un término en la oración prototipo, reemplazando el término con un lado derecho que puede derivarse de esa regla. El primer paso usa la Regla 1 para reemplazar Sentencia. El segundo usa la Regla 2 para reemplazar el Asunto. El tercero reemplaza Objeto usando la Regla 5, mientras que el paso final reescribe Modificador con adjetivo de acuerdo con la Regla 6. En este punto, la oración prototipo generada por la derivación coincide con el flujo de palabras categorizadas producidas por el escáner.

La derivación prueba que la oración "Los compiladores son objetos diseñados". pertenece al lenguaje descrito por las Reglas 1 a 6. La oración es Gramaticalmente correcta. El proceso de encontrar derivaciones automáticamente es llamado análisis.

Una oración gramaticalmente correcta puede no tener sentido. Por ejemplo, la rica oración "Las rocas son vegetales verdes" tiene las mismas partes del discurso en el mismo orden que "Los compiladores son objetos diseñados", pero no tiene significado. Para comprender la diferencia entre estas dos oraciones se requiere conocimiento contextual sobre sistemas de software, rocas y vegetales. Los modelos semánticos que los compiladores utilizan para razonar sobre el lenguaje de programación. la pasada del compilador que comprueba la coherencia de tipos usos de nombres en el programa de entrada Los indicadores son más simples que los modelos necesarios para comprender el lenguaje natural. Un compilador crea modelos matemáticos que detectan tipos específicos de inconsistencia en un programa. Los compiladores comprueban la coherencia del tipo; por ejemplo, la expresión

$a \leftarrow a \times 2 \times b \times c \times d$

pueden estar bien formados sintácticamente, pero si byd son cadenas de caracteres, la oración aún podría no ser válida. Los compiladores también verifican la coherencia del número en situaciones específicas; por ejemplo, una referencia de matriz debe tener el mismo número de dimensiones que el rango declarado de la matriz y un procedimiento La llamada debe especificar el mismo número de argumentos que la definición del procedimiento. El capítulo 4 explora algunos de los problemas que surgen en el tipo basado en compilador comprobación y elaboración semántica.

Representaciones intermedias

El último problema que se maneja en la interfaz de un compilador es la generación de una forma ir del código. Los compiladores usan una variedad de diferentes tipos

de ir, según el idioma de origen, el idioma de destino y la traducción específica $t0 \leftarrow a \times 2$

$t1 \leftarrow t0 \times b$

$t2 \leftarrow t1 \times c$

$t3 \leftarrow t2 \times d$

$a \leftarrow t3$

Algunos irs representan el programa como un gráfico. Otros se asemejan a un programa de código ensamblador secuencial. El código en el margen muestra cómo podría verse nuestra expresión de ejemplo en un nivel bajo, ir secuencial. El capítulo 5 presenta una descripción general de la variedad de tipos de IR que utilizan los compiladores.

Para cada construcción del lenguaje fuente, el compilador necesita una estrategia sobre cómo implementará esa construcción en la forma ir del código. Opciones específicas afectan la capacidad del compilador para transformar y mejorar el código. Por lo tanto, nosotros dedicamos dos capítulos a los problemas que surgen en la generación de ir para código fuente constructos. Los vínculos de procedimientos son, a la vez, una fuente de ineficiencia en el código final y el pegamento fundamental que une diferentes archivos fuente en una aplicación. Por lo tanto, dedicamos el capítulo 6 a las cuestiones que rodean llamadas a procedimiento. El capítulo 7 presenta estrategias de implementación para la mayoría de las demás construcciones de lenguaje de programación

El optimizador

Cuando el front-end emite ir para el programa de entrada, maneja las declaraciones uno a la vez, en el orden en que se encuentran. Así, la ir inicial. El programa contiene estrategias generales de implementación que funcionarán en cualquier contexto circundante que el compilador podría generar. En tiempo de ejecución, el código se ejecutará en un contexto más restringido y predecible. El optimizador analiza la forma del código para descubrir hechos sobre ese contexto y usa ese conocimiento contextual para reescribir el código para que calcule el mismo responder de una manera más eficiente.

La eficiencia puede tener muchos significados. La noción clásica de optimización es para reducir el tiempo de ejecución de la aplicación. En otros contextos, el optimizador podría intentar reducir el tamaño del código compilado u otras propiedades como la energía que consume el procesador al evaluar el código. Todos estos las estrategias apuntan a la eficiencia. Volviendo a nuestro ejemplo, considérela en el contexto que se muestra en la Figura 1.2a. La declaración ocurre dentro de un bucle. De los valores que utiliza, solo un y d cambio dentro del bucle. Los valores de 2, byc son invariantes en el círculo. Si el optimizador descubre este

hecho, puede reescribir el código como se muestra en Figura 1.2b. En esta versión, se ha reducido el número de multiplicaciones

de $4 \cdot n$ a $2 \cdot n + 2$. Para $n > 1$, el bucle reescrito debería ejecutarse más rápido. Esto El tipo de optimización se analiza en los capítulos 8, 9 y 10.

Análisis

La mayoría de las optimizaciones consisten en un análisis y una transformación. El análisis determina dónde el compilador puede aplicar la técnica de forma segura y rentable. **Análisis del flujo de datos** Los compiladores utilizan varios tipos de análisis para respaldar las transformaciones. **Datos** una forma de razonamiento en tiempo de compilación sobre el flujo de valores en tiempo de ejecución **razones de análisis de flujo**, en tiempo de compilación, sobre el flujo de valores en tiempo de ejecución. Los analizadores de flujo de datos suelen resolver un sistema de ecuaciones de conjuntos simultáneos que se derivan de la estructura del código que se está traduciendo. **Dependencia** El análisis utiliza pruebas teóricas de números para razonar sobre los valores que se pueden

Transformación

Para mejorar el código, el compilador debe ir más allá de analizarlo. El compilador debe usar los resultados del análisis para reescribir el código en un formato más forma eficiente. Se han inventado innumerables transformaciones para mejorar los requisitos de tiempo o espacio del código ejecutable. Algunos, como descubrir cálculos invariantes de bucle y moverlos a ejecutados con menos frecuencia ubicaciones, mejorar el tiempo de ejecución del programa. Otros hacen el código Mas Compacto. Las transformaciones varían en su efecto, el alcance sobre el cual operan, y el análisis requerido para apoyarlos. La literatura sobre las transformaciones es rica; el tema es lo suficientemente grande y profundo para merecen uno o más libros separados. El capítulo 10 cubre el tema de escalar transformaciones, es decir, transformaciones destinadas a mejorar el rendimiento del código en un solo procesador. Presenta una taxonomía para organizar el tema y llena esa taxonomía con ejemplos.

El back-end

El back-end del compilador atraviesa la forma ir del código y emite código para la máquina de destino. Selecciona las operaciones de la máquina objetivo para implementar cada operación de ir. Elige un orden en el que se ejecutarán las operaciones eficientemente. Decide qué valores residirán en los registros y qué valores residirá en la memoria e inserta código para hacer cumplir esas decisiones

Actividades en equipo:

2) Multi-Level Intermediate Representation (MLIR)

- **MLIR**

Es un proyecto que define una representación intermedia (IR) común que unifica la infraestructura necesaria para ejecutar modelos de aprendizaje automático de alto rendimiento en Tensor Flow y en marcos de trabajo de AA similares.

Este proyecto incluye la aplicación de técnicas de HPC (computación de alto rendimiento), junto con la integración de algoritmos de búsqueda, como el aprendizaje por refuerzo. El objetivo de MLIR es reducir el costo para incorporar hardware nuevo y mejorar la usabilidad para los usuarios de Tensor Flow.

MLIR tiene algunas propiedades interesantes (forma SSA, operaciones de dialecto, regiones) que hacen que el trabajo sea mucho más fácil de cambiar. Pero lo que es más importante, el IR tiene un flujo de control explícito (CFG), que es importante para rastrear de dónde provienen las variables, pasan a través de todas las combinaciones de rutas.

Para inferir tipos y verificar la seguridad, esto es fundamental para asegurarse de que el código no pueda llegar a un estado desconocido a través de al menos una de las rutas posibles. Entonces, la razón principal por la que elegimos MLIR para ser nuestra representación es para que podamos hacer nuestra inferencia de tipo más fácilmente.

La segunda razón es que MLIR nos permite mezclar cualquier número de dialectos. Por lo tanto, podemos reducir el AST a una mezcla de dialecto de Verona y otros dialectos estándar, y los pases que solo pueden ver las operaciones de Verona ignorarán a los demás y viceversa.

También nos permite bajar parcialmente partes del dialecto a otros dialectos sin tener que convertir todo. Esto mantiene el código limpio (pasadas cortas y directas) y nos permite construir lentamente más información, sin tener que ejecutar una pasada de análisis enorme seguida de una pasada de transformación enorme, solo para perder información en el medio.

Un beneficio inesperado de MLIR fue que tiene soporte nativo para operaciones opacas, es decir. operaciones similares a llamadas a funciones que no necesitan definirse en ninguna parte.

MLIR no tiene una representación de cadena nativa y no hay una forma sensata de representar todos los tipos de cadenas en los tipos existentes.

F) Actividades semana mayo 3-7, 2021

Actividades individuales

- 1) "Chapter Summary and Perspective" del libro "Engineering a Compiler" de Cooper y Torczon (2012).

2.7 RESUMEN Y PERSPECTIVA DEL CAPÍTULO

El uso generalizado de expresiones regulares para buscar y escanear es una de las historias de éxito de la informática moderna. Estas ideas fueron desarrollando como una parte temprana de la teoría de lenguajes formales y autómatas. Se aplican de forma rutinaria en herramientas que van desde editores de texto hasta filtrado web. Motores a compiladores como un medio de especificar de manera concisa grupos de cadenas que resultan ser lenguajes regulares. Siempre que una colección finita de palabras debe ser reconocidos, los reconocedores basados en dfa merecen una seria consideración.

La teoría de expresiones regulares y autómatas finitos ha desarrollado técnicas que permiten el reconocimiento de lenguajes regulares en tiempo proporcional a la longitud del flujo de entrada. Técnicas para la derivación automática dudas de res y para la minimización de dfa han permitido la construcción de herramientas robustas que generan reconocedores basados en DFA. Tanto los escáneres generados como los hechos a mano se utilizan en compiladores modernos muy respetados. En cualquier caso, una implementación cuidadosa debe ejecutarse en el tiempo proporcional a la duración del flujo de entrada, con una pequeña sobrecarga por carácter. de exploración crece linealmente con el número de caracteres, y la constante. Los costos son bajos, lo que empuja el análisis léxico del analizador a un el escáner redujo el costo de compilación. El advenimiento de técnicas de análisis eficientes debilitó este argumento, pero la práctica de construir escáneres persiste porque proporciona una clara separación de preocupaciones entre la estructura léxica y estructura sintáctica.

Debido a que la construcción del escáner juega un papel pequeño en la construcción de un compilador real, hemos tratado de que este capítulo sea breve. Por tanto, el capítulo omite muchos teoremas sobre lenguajes regulares y autómatas finitos que el lector ambicioso podría disfrutar. Los muchos buenos textos sobre este tema pueden proporcionar una descripción mucho más profunda tratamiento de autómatas finitos y expresiones regulares, y sus muchas propiedades

Kleene [224] estableció la equivalencia de res y fas. Tanto el Kleene El cierre y el algoritmo de dfa to re llevan su nombre. McNaughton y Yamada mostró una construcción que se relacionares

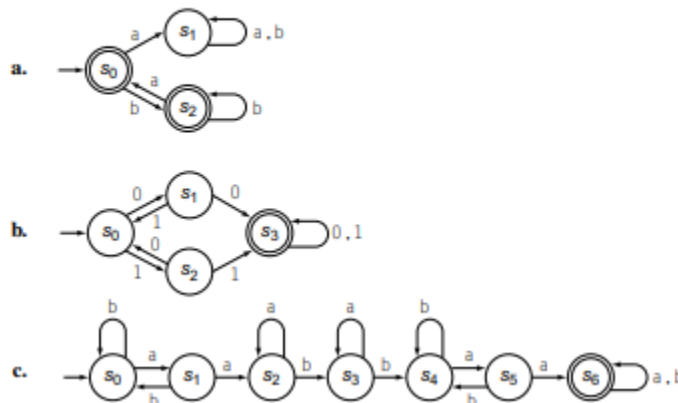
La construcción que se muestra en este capítulo sigue el modelo del trabajo de Thompson que fue motivado por la implementación de un comando de búsqueda textual para un editor de texto temprano. Johnson describe la primera aplicación de esta tecnología a automatizar la construcción del escáner. La construcción del subconjunto se deriva de Rabin y Scott. El algoritmo de minimización de dfa en la Sección 2.4.4 se debe a Hopcroft Ha encontrado aplicación para muchos problemas diferentes, incluida la detección cuando dos variables de programa siempre tienen el mismo valor

La idea de generar código en lugar de tablas, para producir un código directo escáner, parece tener su origen en el trabajo de Waite y Heuring Informan un factor de cinco de mejora con respecto a las implementaciones basadas en tablas. Ngassam y col. describir experimentos que caracterizan las posibles aceleraciones en escáneres codificados a mano. Varios autores han examinado las compensaciones en implementación del escáner. Jones [208] aboga por la codificación directa, pero defiende un enfoque estructurado para controlar el flujo en lugar del código espagueti que se muestra en la Sección 2.5.2. Brouwer y col. comparar la velocidad de 12 implementaciones de escaneo diferentes; descubrieron un factor de 70 diferencia entre los implementaciones más rápidas y más lentas.

La técnica alternativa de minimización de dfa presentada en la Sección 2.6.2 fue descrito por Brzozowski en 1962 [60]. Varios autores han comparado las técnicas de minimización de dfa y su rendimiento. Muchos los autores han examinado la construcción y minimización de acíclicos

■ EXERCISES

1. Describe informally the languages accepted by the following FAS:



3.7 RESUMEN Y PERSPECTIVA

Casi todos los compiladores contienen un analizador. Durante muchos años, el análisis fue un tema de gran interés. Esto condujo al desarrollo de muchas técnicas para construir analizadores sintácticos eficientes. La familia de gramáticas $lr(1)$ incluye todas las gramáticas libres de contexto que se pueden analizar de forma determinista. Las herramientas producen analizadores sintácticos eficientes con pruebas sólidas propiedades de detección de errores. Esta combinación de características, junto con la amplia disponibilidad de generadores de analizadores sintácticos para $lr(1)$, $lrlr(1)$ y $slr(1)$ gramáticas, ha disminuido el interés en otras técnicas de análisis automático como analizadores de precedencia de operadores.

Los analizadores de descendencia recursiva descendente tienen su propio conjunto de ventajas. Ellos son, posiblemente, los analizadores codificados a mano más fáciles de construir. Ellos proveen excelentes oportunidades para detectar y reparar errores de sintaxis. Son eficientes; de hecho, un analizador sintáctico descendente recursivo de arriba hacia abajo bien construido puede ser más rápido que un analizador $lr(1)$ basado en tablas. (El esquema de codificación directa para $lr(1)$ puede superar esta ventaja de velocidad.) En un analizador sintáctico descendente recursivo de arriba hacia abajo, el

El redactor del compilador puede afinar más fácilmente las ambigüedades en el lenguaje fuente. que pueden causar problemas a un analizador $lr(1)$, como un idioma en el que la palabra clave los nombres pueden aparecer como identificadores. Un redactor de compiladores que quiere construir un analizador codificado a mano, por el motivo que sea, se recomienda utilizar el método de descenso recursivo.

Al elegir entre las gramáticas $lr(1)$ y $ll(1)$, la elección se convierte en una de las siguientes: herramientas disponibles. En la práctica, pocas construcciones de lenguaje de programación, si es que hay alguna, caen en la brecha entre las gramáticas $lr(1)$ y $ll(1)$ gramáticas. Por lo tanto, comenzar con un generador de analizador sintáctico disponible es siempre mejor que implementar un generador de analizador desde cero.

Hay disponibles algoritmos de análisis más generales. En la práctica, sin embargo, las restricciones impuestas a las gramáticas libres de contexto por las clases $lr(1)$ y $ll(1)$ no causan problemas para la mayoría de los lenguajes de programación.

■ NOTAS DEL CAPÍTULO

Los primeros compiladores utilizaban analizadores sintácticos codificados a mano. La riqueza sintáctica de Algol 60 desafió a los primeros escritores de compiladores. Intentaron una variedad de esquemas para analizar el lenguaje; Randell y Russell ofrecen una visión general fascinante de los métodos utilizados en una variedad de compiladores de Algol. Irons fue uno de los primeros en separar la noción de sintaxis de la traducción. Lucas parece haber introducido la noción de descendencia recursiva análisis sintáctico. Conway aplica ideas similares a un eficiente compilador para cobol. Las ideas detrás del análisis sintáctico LL y LR aparecieron en la década de 1960. Lewis y Stearns introdujo LL(k) gramáticas; Rosenkrantz y Stearns describieron sus propiedades con mayor profundidad. Foster desarrolló un algoritmo para transformar una gramática en forma LL(1). Wood formalizó la noción de factorización izquierda una gramática y exploró las cuestiones teóricas implicadas en la transformación de una gramática a la forma LL(1).

Knuth expuso la teoría detrás del análisis sintáctico de LR(1). DeRemer y otros desarrollaron técnicas, los algoritmos de construcción de tablas SLR y LALR, que hizo que el uso de generadores de analizador LR sea práctico en la memoria limitada computadoras del día. Waite y Goos describen una técnica para eliminar automáticamente las producciones inútiles durante el algoritmo de construcción de la tabla LR(1). Penello sugirió la codificación directa de las tablas en código ejecutable. Aho y Ullman es una referencia definitiva tanto en el análisis sintáctico de LL como de LR. Bill Waite proporcionó el ejemplo de gramática en ejercicio 3.7.

Aparecieron varios algoritmos para analizar gramáticas arbitrarias libres de contexto en la década de 1960 y principios de la de 1970. Algoritmos de Cocke y Schwartz, Younger, Kasami y Earley tenían una complejidad computacional similar. El algoritmo de Earley merece una mención especial debido a su similitud con el algoritmo de construcción de tablas LR(1). Algoritmo de Earley deriva el conjunto de posibles estados de análisis en tiempo de análisis, en lugar de en tiempo de ejecución, donde las técnicas LR(1) los calculan previamente en un generador de analizador sintáctico. A partir de una vista de alto nivel, los algoritmos LR(1) pueden aparecer como una optimización natural del algoritmo de Earley.

4. The following grammar is not suitable for a top-down predictive parser. Identify the problem and correct it by rewriting the grammar. Show that your new grammar satisfies the LL(1) condition.

$$\begin{array}{lll} L \rightarrow R a & R \rightarrow a b a & Q \rightarrow b b c \\ | Q b a & | c a b a & | b c \\ & | R b c & \end{array}$$

5. Consider the following grammar:

$$\begin{array}{ll} A \rightarrow B a & C \rightarrow c B \\ B \rightarrow d a b & | A c \\ | C b & \end{array}$$

4.6 RESUMEN Y PERSPECTIVA

En los capítulos 2 y 3, vimos que gran parte del trabajo en el frente de un compilador final se puede automatizar. Las expresiones regulares funcionan bien para el análisis léxico. Las gramáticas libres de contexto funcionan bien para el análisis de sintaxis. En este capítulo, Examinamos dos formas de realizar análisis sensibles al contexto: el formalismo gramatical de atributos y un enfoque ad hoc. Para el análisis sensible al contexto, a diferencia del escaneo y el análisis sintáctico, el formalismo no ha desplazado al acercarse.

El enfoque formal, que utiliza gramáticas de atributos, ofrece la esperanza de escribir especificaciones de alto nivel que produzcan ejecutables razonablemente eficientes. Si bien las gramáticas de atributos no son la solución a todos los problemas del análisis sensible al contexto, han encontrado aplicación en varios dominios, que van desde probadores de teoremas hasta análisis de programas. Para problemas en los que el

El flujo de atributos es principalmente local, las gramáticas de atributos funcionan bien. Problemas que se puede formular completamente en términos de un tipo de atributo, ya sea heredado o sintetizados, a menudo producen soluciones limpias e intuitivas cuando se presentan como atributo gramáticas. Cuando el problema de dirigir el flujo de atributos alrededor del árbol con reglas de copia llega a dominar la gramática, probablemente es hora de salir del paradigma funcional de las gramáticas de atributos e introducir un repositorio central de hechos.

La técnica ad hoc, traducción dirigida por sintaxis, integra recortes arbitrarios de código en el analizador y permite al analizador secuenciar las acciones y pasar valores entre ellos. Este enfoque ha sido ampliamente adoptado debido a su flexibilidad y su inclusión en la mayoría de los sistemas generadores de analizadores sintácticos. El enfoque evita los problemas prácticos que surgen de los atributos no locales flujo y de la necesidad de gestionar el almacenamiento de atributos. Los valores fluyen en una dirección junto con la representación interna del analizador de su estado (sintetizados valores para analizadores ascendentes y heredados para analizadores descendentes). Estas Los esquemas utilizan estructuras de datos globales para pasar información en la otra dirección. y para manejar el flujo de atributos no locales.

En la práctica, el redactor del compilador a menudo intenta resolver varios problemas en una vez, como construir una representación intermedia, inferir tipos y Asignación de ubicaciones de almacenamiento. Esto tiende a crear importantes flujos de atributos. en ambas direcciones, empujando al implementador hacia una solución que utiliza algún repositorio central de hechos, como una tabla de símbolos.

La justificación para resolver muchos problemas en una sola pasada suele ser la eficiencia del tiempo de compilación. Sin embargo, resolver los problemas en pasadas separadas puede a menudo producen soluciones que son más fáciles de entender, implementar y mantener.

Este capítulo presentó las ideas detrás de los sistemas de tipos como un ejemplo de tipo de análisis sensible al contexto que debe realizar un compilador. El estudio de la teoría de tipos y el diseño de sistemas de tipos es una actividad académica significativa con una profunda literatura propia. Este capítulo arañó la superficie de la inferencia de tipos y la verificación de tipos, pero un tratamiento más profundo de estos temas está más allá del alcance de este texto. En la práctica, el redactor del compilador debe estudiar a fondo el sistema de tipos del lenguaje fuente y diseñar la implementación de inferencia de tipo y comprobación de tipo con cuidado. Los consejos de este capítulo son un comienzo, pero una implementación realista requiere más estudio.

■ NOTAS DEL CAPÍTULO

Los sistemas de tipos han sido una parte integral de los lenguajes de programación desde el compilador original de fortran. Mientras que los primeros sistemas de tipos reflejaban los recursos de la máquina subyacente, pronto niveles más profundos de abstracción aparecieron en sistemas de tipos para lenguajes como Algol 68 y Simula 67.

La teoría de los sistemas de tipos se ha estudiado activamente durante décadas, produciendo una serie de lenguajes que incorporan principios importantes. Éstas incluyen Russell (polimorfismo paramétrico), clu (tipos de datos abstractos), Smalltalk (subtipificación por herencia) y ml (minucioso y tratamiento completo de tipos como objetos de primera clase). Cardelli ha escrito una excelente descripción de los sistemas de tipos. La comunidad apl produjo una serie de artículos clásicos que trataban sobre técnicas para eliminar el tiempo de ejecución chequeos

Las gramáticas de atributos, como muchas ideas en informática, fueron propuestas por primera vez por Knuth. La literatura sobre gramáticas de atributos se ha centrado sobre evaluadores, sobre pruebas de circularidad y sobre aplicaciones de gramáticas de atributos. Las gramáticas de atributos han servido como base para varios sistemas exitosos, incluido el compilador Pascal de Intel para el 80286, el sintetizador de programa Cornell y el sintetizador Generador

La traducción ad hoc dirigida por sintaxis siempre ha sido parte del desarrollo de analizadores reales. Irons describió las ideas básicas detrás de la traducción dirigida por sintaxis para separar las acciones de un analizador de la descripción de su sintaxis.

Sin lugar a dudas, se utilizaron las mismas ideas básicas en la precedencia codificada a mano. analizadores sintácticos. El estilo de escribir acciones dirigidas por la sintaxis que describimos fue introducido por Johnson en Yacc. Se ha llevado la misma notación avanzar a sistemas más recientes, incluido el bisonte del proyecto Gnu.

Production	Evaluation Rules
$E_0 \rightarrow E_1 + T$	{ $E_0.nptr \leftarrow mknode(+, E_1.nptr, T.nptr)$ }
$E_0 \rightarrow E_1 - T$	{ $E_0.nptr \leftarrow mknode(-, E_1.nptr, T.nptr)$ }
$E_0 \rightarrow T$	{ $E_0.nptr \leftarrow T.nptr$ }
$T \rightarrow (E)$	{ $T.nptr \leftarrow E.nptr$ }
$T \rightarrow id$	{ $T.nptr \leftarrow mkleaf(id, id.entry)$ }

Actividades en equipo

- 3) Video de presentación de avance del proyecto final del concepto "Multi-Level Intermediate Representation (MLIR)".



¿ PARA QUE SIRVE ?



TENSOR FLOW



VENTAJAS DE USAR TENSOR FLOW

- **COMPILACIÓN SENCILLA DE MODELOS**
- **PRODUCCIÓN DE AA SÓLIDO EN CUALQUIER PARTE**
- **IMPORTANTE EXPERIMENTACIÓN PARA LA INVESTIGACIÓN**

¿QUÉ ES HPC?



BIBLIOGRAFÍA

- PAGINA OFICIAL DE TENSOR FLOW [HTTPS://WWW.TENSORFLOW.ORG/?HL=ES-419](https://www.tensorflow.org/?hl=es-419)
- NETAPP. (2021) ¿QUÉ ES LA COMPUTACION DE ALTO RENDIMIENTO? RECUPERADO DE LA PAGINA OFICIAL: [HTTPS://WWW.NETAPP.COM/ES/DATA-STORAGE/HIGH-PERFORMANCE-COMPUTING/WHAT-IS-HPC/](https://www.netapp.com/es/data-storage/high-performance-computing/what-is-hpc/)
- ORIOl VINyALS, PETE WARDEN, MARTIN WATTENBERG, MARTIN WICKE, YUAN YU, AND XIAOQIANG ZHENG. TENSORFLOW: LARGE-SCALE MACHINE LEARNING ON HETEROGENEOUS SYSTEMS, 2015. SOFTWARE AVAILABLE FROM TENSORFLOW.ORG. RECUPERADO DE: [HTTPS://WWW.TENSORFLOW.ORG/MLIR?HL=ES-419](https://www.tensorflow.org/mlir?hl=es-419)
- MLIR. MULTI-LEVEL IR COMPILER FRAMEWORK HUGO.TECHDOC. THINGSYM. RECUPERADO DE: [HTTPS://MLIR.LLVM.ORG/](https://mlir.llvm.org/)

LIGA DEL VIDEO:

<https://youtu.be/a8A6r57KKEw>

G) Actividades en clase:

Capturas de los ejercicios en clase

Profe:

```
if (x==j)
    z=0;
else
    z=1;
```

Estructura de un token:

<tipo o clase, "cadena">

<w, ">

<k, "if">

Tipos o clases de tokens:

W: espacio en blanco

K: palabra clave

I: identificador

N: número

O: otros tokens (,),[,],>,<,<=,;

Ejemplo: <K,"if">

Los Autobots

Cuanenemi Cuanalo Mario Alberto
Fermín Cruz Erik
Gutiérrez Arellano Rafael
Pérez armas Fausto Isaac

ESTRUCTURA TOKENS

```
if (x==j)
  z=0;
else
  z=1;
```

Estructura de un token:
<tipo o clase, "cadena">

Tipos o clases de tokens:

W: .espacio en blanco

K: palabra clave

I: identificador

N: número

O: otros tokens (,),[,],>,<,<=,>=

Ejemplo: <K,"if">

<k,"if">

<W, " ">

<O, "(">

<I, "x">

<O, "==">

<I, "j">

<O, ")">

<W, " ">

<I, "z">

<O, "=">

<N, "0">

<O, ";">

<W, "">

<k,"else">

<W, " ">

<I, z>

Profe: Ejercicios: Con el alfabeto $\Sigma = \{0,1\}$

1) Con la expresión regular: $(1+0)^1$ escribe 5 cadenas válidas

R= No existen 5 cadenas válidas de esta expresión regular, las únicas cadenas válidas son: 11,01

2) Con la expresión regular: $(1+0)^*1$ escribe 5 cadenas válidas

3) $0^* + 1^*$

4) $(0^*1)^0$

5) $1(1^*010^*)0^*$

6) $(1+0)^+(0+1)^*+(0^*1^*)$

Práctica 1: Comprobar las cadenas propuestas en las 6 expresiones regulares empleando jFLAP

Tarea: Resolver el ejercicio 1 de la página 80 del libro "Engineering a Compiler"

$a^* = \epsilon + a + aa + aaa + aaaa + aaaaa + aaaaaa + aaaaaaa + aaaaaaaaa + \dots$

Definición formal de la "Cerradura de Kleene"

**Los
Autobots**

Cuanenemi Cuanalo Mario Alberto
Fermín Cruz Erik
Gutiérrez Arellano Rafael
Pérez armas Fausto Isaac

En términos de concatenación: ¿ $a \cdot a = aa$?
No es lo mismo $a = aa$

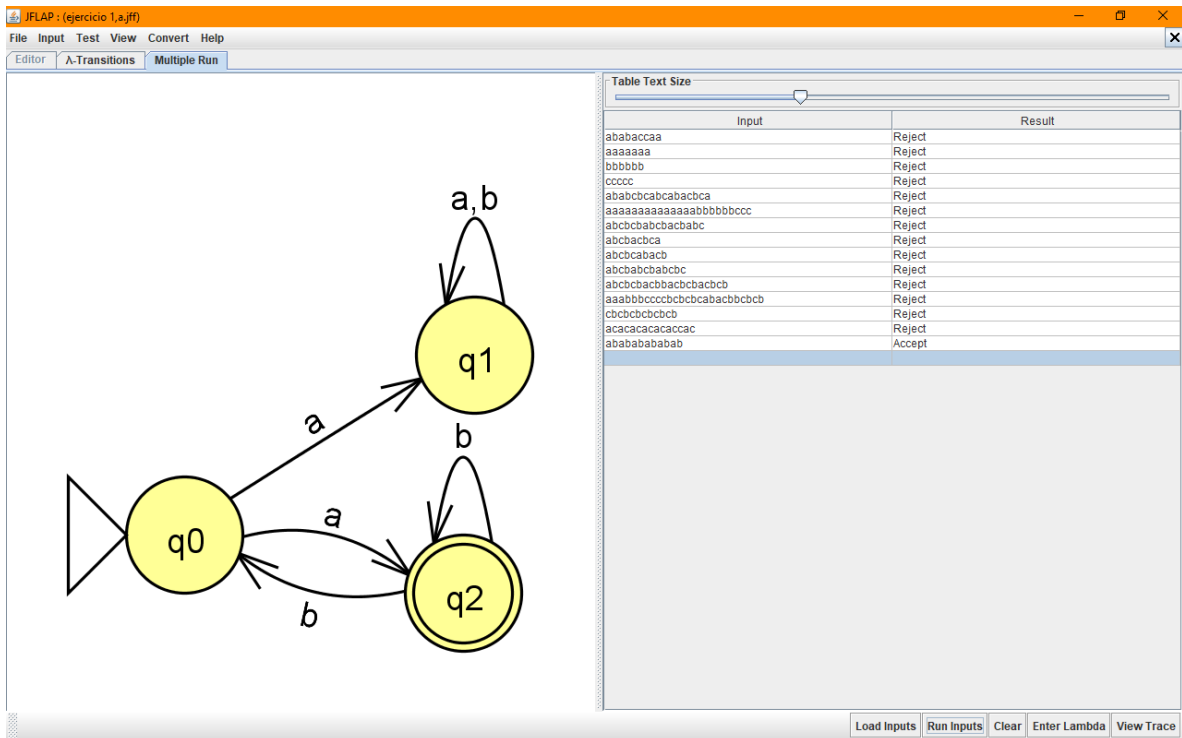
Ejercicios: Con el alfabeto $\Sigma = \{0,1\}$

1) Con la expresión regular: $(1+0)^1$ escribe 5 cadenas válidas:

- | | |
|---------------|-------------|
| 1.- $(1+0)^0$ | 1.- 1010101 |
| 2.- $(0+1)^1$ | 2.- 0001010 |
| 3.- $(0+0)^1$ | 3.- 01001 |
| 4.- $(1+1)^0$ | 4.- 1111111 |
| 5.- $(0+1)^0$ | 5.- 0000000 |



- 1.- $(1+0)^1$
101
110
- 2.- 0^*1
0001
01
010101
001
0011
- 3.- $(0^*1)^0$
0010
010
0100
010010
01010
- 4.- $1(1^*010^*)0^*$
100100
1010
10101010
1010010
10100
- 6.- $(1+0)^1 + (0+1)^* + (0^*1)^*$
100101
10010101
100101010
1010010
1010



JFLAP : (ejercicio 1.b.jp)

File Input Test View Convert Help

Editor Multiple Run

Table Text Size

Input	Result
ababacca	Reject
aaaaaaa	Reject
bbbbbb	Reject
cccc	Reject
ababcbcabcbacba	Reject
aaaaaaaaaaaaabbbbbb	Reject
abcbcbabcbabcb	Reject
abcbabcb	Reject
abcbabcb	Reject
abcbabcbabcb	Reject
abcbabcbabcbabcb	Reject
aaabbbccccbcbcbabcbcbcb	Reject
cbcbcbcbcb	Reject
acacacacacac	Reject
ababababab	Reject

Load Inputs Run Inputs Clear Enter Lambda View Trace

JFLAP : (ejercicio 1.c.jp)

File Input Test View Convert Help

Editor Multiple Run

Table Text Size

Input	Result
ababacca	Reject
aaaaaaa	Reject
bbbbbb	Reject
cccc	Reject
ababcbcabcbacba	Reject
aaaaaaaaaaaaabbbbbb	Reject
abcbcbabcbabcb	Reject
abcbabcb	Reject
abcbabcb	Reject
abcbabcbabcb	Reject
abcbabcbabcbabcb	Reject
aaabbbccccbcbcbabcbcbcb	Reject
cbcbcbcbcb	Reject
acacacacacac	Reject
ababababab	Reject

Load Inputs Run Inputs Clear Enter Lambda View Trace