Qwen

De la Generació a la Fiabilitat: Un Model Integrat per a la Creació i Verificació de Codis IEC 61131-3

Panorama de la Generació Automàtica de Codis IEC 61131-3

La generació automàtica de codi per a controladors lògics programables (PLC) ha evolucionat significativament, passant de metodologies model-bases tradicionals a sistemes impulsats per Intel·ligència Artificial (IA). Aquesta transició no només busca accelerar el cicle de vida del desenvolupament de software, sinó també millorar la qualitat i fiabilitat dels programes industrials [29]. El llenguatge Structured Control Language (sCL), una implementació específica de Structured Text (ST) definida en l'estàndard IEC 61131-3, és un objectiu central d'aquests nous enfocaments [16] [32]. L'estàndard IEC 61131-3 és la pedra angular de la programació industrial, sent adoptat universalment per a garantir la portabilitat i interoperabilitat dels programes entre diferents fabricants [10] [36]. Part 3 d'aquest estàndard especifica detalladament la sintaxi i semàntica d'una suite de cinquè llenguatges de programació per a controladors lògics [12] [50].

Les aproximacions tradicionals a la generació automàtica es basen principalment en el Model-Driven Development (MBD). Aquesta filosofia utilitza models abstractes, sovint gràfics o matemàtics, per generar codi objectiu de manera directa i fiable. Una de les eines més representatives en aquest camp és el Simulink PLC Coder de MathWorks, que permet generar codi IEC 61131-3 Structured Text directament des d'un model de Simulink [60] [61]. Aquesta metodologia integra-se perfectament en el flux de treball de Model-Based Design, simplificant la transició des del model d'algorisme fins a la implementació sobre el PLC [72]. Altres investigacions han explorat la generació automàtica a partir de diversos tipus de models formals, com ara UML (Llenguatge Unificat de Modelatge) [63], Timed MPSG (un model d'estats finits extendit) [7], o models de processos orientats coneguts com poST [65] [67]. Aquestes aproximacions, encara que robustes i ben establertes, sovint requereixen eines especialitzades i poden presentar una curva de pujada elevada per als usuaris.

Una corrent de recerca molt activa i emergent és l'aplicació de Models Largs de Llenguatge (LLM) per a la generació de codi IEC 61131-3. Diverses publicacions científiques destaquen el potencial d'aquestes tecnologies. El marc de treball Agents4PLC, per exemple, introdueix un sistema que no només genera codi SCL de forma automàtica, sinó que també incorpora una fase de verificació a nivell de codi basada en IA, creant un cicle tancat i iteratiu [37] [68]. Un altre projecte destacat és LLM4PLC, que proposa una línia de producció guiada per l'usuari per refinar iterativament el codi generat per LLMs. El seu mètode va demostrar una millora significativa, augmentant la taxa d'èxit de generació del 47% a un 72% i obtenint una qualitat de codi molt superior segons una enquesta d'experts [6] [38]. El projecte Spec2Control va anar un pas enllà, generant directament diagrames de blocs de funció (FBD), un dels llenguatges gràfics de l'IEC 61131-3, a partir de requisits descrits en llenguatge natural mitjançant un flux de treball complet basat en LLMs [69]. Aquestes tendències indiquen una direcció cap a una major abstracció i capacitat creativa en la generació de codi.

No obstant això, un dels reptes més importants en aquest domini és assegurar que el sistema generador sigui conscient del proveïdor ("vendor-aware"). Tot i que l'estàndard IEC 61131-3 intenta unificar la sintaxi i la semàntica, els fabricants de PLCs com Siemens (amb la seva implementació de Structured Control Language per a TIA Portal [30]) o CODESYS [15] hi afegeixen extensions, funcions especialitzades i comportaments únics [2]. Un sistema de generació eficaç per a entorns industrials reals ha de poder adaptar-se a aquesta variabilitat per generar codi que no només sigui correcte respecte a l'estàndard general, sinó que també faci servir de manera optimitzada les capacitats específiques del hardware i software del proveïdor seleccionat. El projecte AutoPLC ja està treballant en aquesta direcció, reconeixent que les implementacions reals de ST són riques en extensions de cada proveïdor [2] [15]. La capacitat de gestionar aquest context de proveïdor esdevé, per tant, un factor clau de diferenciació i valor pràctic per a qualsevol eina de generació automàtica moderna.

Metodologia de Generació	Descripció	Exemples d'Eines/ Projectes	Avantatges	Inconvenients
Model-Driven Development (MBD)	Genera codi IEC 61131-3 a partir de models gràfics o matemàtics (p. ex., Simulink, UML) [61] [63].	Simulink PLC Coder [60] [61], MBD tools for UML [63]	Alta fiabilitat, integració amb fluxos de treball de modelatge avançat, facilita la simulació pre-codi.	Requereix eines especialitzades, pot tenir una curva de pujada elevada, menys flexibilitat per a codi ad-hoc.
Intel·ligència Artificial (LLMs)	Utilitza Models Largs de Llenguatge per generar codi a partir de textos o diagrames. Pot ser textual o gràfic [37] [69].	Agents4PLC [37], LLM4PLC [38], Spec2Control [69], AutoPLC [2]	Gran capacitat creativa, adaptabilitat, pot reduir dràsticament el temps de desenvolupament.	Qualitat del codi variable, necessita refinament iteratiu, pot generar codi ineficient o incorrecte sense verificació.
Generació Basada en Regles/ Formalismes Especialitzats	Transforma models específics (p. ex., de processos) en codi IEC 61131-3 [7] [65].	Frameworks basats en Timed MPSG [7], poST [65] [67]	Adreça casos d'ús específics amb rigor formal, pot generar codi altament optimitzat per a aquell paradigma.	Menys generalistes, aplicable a dominis més limitats.

En resum, el panorama de la generació automàtica de codi IEC 61131-3 és ric i divers. Les metodologies basades en models ofereixen una base sòlida i fiable, mentre que les aproximacions basades en IA obrin noves possibilitats per a la productivitat i la creativitat. No obstant això, cap d'aquestes aproximacions és completa per si sola. La generació de codi de qualitat per a entorns industrials complexos exigeix una atenció especial a la compatibilitat amb les implementacions específiques dels proveïdors, un aspecte que defineix el futur de les eines de generació modernes.

Mètodes Avançats de Verificació i Validació de Programes

La fiabilitat del codi programable en controladors lògics és primordial en sistemes d'automatització, on fallides poden tenir conseqüències severes [29]. Per aquest motiu, la verificació i validació del codi sCL/ST generat és tan important com el procés de generació en si mateix. L'estàndard IEC 61131-3 no només defineix els llenguatges, sinó que també promou la conformitat amb bones pràctiques per a millorar la qualitat del software [27]. Els mètodes de verificació es troben en un espectre que va des de l'anàlisi estàtic accessible fins a la verificació formal, una disciplina acadèmica però de gran impacte pràctic per a sistemes crítics.

L'anàlisi estàtic és el primer nivell de verificació i ja està implementat en molts entorns de desenvolupament integrat (IDEs) per a PLCs [41]. Aquesta tècnica analitza el codi font

sense executar-lo, detectant una àmplia gamma de problemes potencials. Les funcionalitats típiques inclouen la comprovació de la sintaxi respecte al llenguatge IEC 61131-3 [41], l'aplicació de regles personalitzades sobre convencions de nomenclatura, i la mesura de metriques de qualitat del codi [54]. Una de les metriques més útils és la complexitat ciclomàtica, que quantifica la complexitat d'un programa i serveix com a indicador de la seva probabilitat de contenir errors; eines com CoDeSys Static Analysis poden calcular-la [79]. A més, els analitzadors estàtics poden identificar problemes com variables no inicialitzades, codi mort o funcions excessivament complexes [24] [58]. Plataformes com CODESYS i TwinCAT incorporen eina d'anàlisi estàtica per ajudar els programadors a escriure codi més llegible i defectuós [41] [55] [56] [57]. També existeixen eines externes com l'IEC-Checker, que valida el codi ST contra les regles de PLCopen per detectar vulnerabilitats recurrents [24].

Quan l'anàlisi estàtic no és suficient i es necessita una garantia més alta de correcció, es recorre a la verificació formal. Aquest conjunt de tècniques utilitza mètodes matemàtics per provar si un programa compleix amb certes especificacions. Una de les tècniques principals és el model checking, que transforma el programa en un model matemàtic i verifica exhaustivament si aquest model satisfà propietats lògiques desitjades [70]. Diverses eines estan desenvolupades específicament per a la verificació de programes IEC 61131-3. **PLCverif** és una eina dissenyada per fer accessible el model checking als desenvolupadors de PLC, permetent-los verificar propietats complexes del seu codi [9] [28]. Un altre recurs important és **K-ST**, que proporciona una semàntica formal executable de Structured Text [29] [31]. Aquesta semàntica actua com a referència precisa per avaluar el comportament del codi generat i pot ser utilitzada per validar-lo contra especificacions formalitzades [40] [73] [74]. Alguns marcs de treball també integren eina de model checking externes com nuXmv o Uppaal per realitzar la verificació [28] [77]. A més, existeixen altres aproximacions com la traducció automàtica del codi a màquines B per a la seva verificació formal [33] o l'aplicació de normatives de seguretat funcional com la IEC 61508 per a la validació de sistemes segurs [14] [53].

Una àrea de recerca activa és la validació basada en especificacions, que busca superar la dependència d'especificacions manuales explícites. Un mètode innovador consisteix a minar especificacions a partir del propi codi del programa, permetent una verificació posterior més informada [3]. El projecte LLM4PLC combina aquest concepte amb la generació de codi, construint una línia de producció que utilitza coneixements d'anàlisi estàtica per verificar el codi generat per LLM [22]. Aquest enfocament híbrid representa una direcció potent, ja que combina la creativitat de la generació automàtica amb les garanties de la verificació sistemàtica.

Nivell de Verificació	Mètode Principal	Objectiu	Eines i Tècniques Clau	Aplicabilitat
Anàlisi Estàtic	Anàlisi del codi font sense executar-lo.	Detectar errors comuns, problemes de qualitat i desviacions de bones pràctiques.	CODESYS Static Analysis [54], TwinCAT Static Analysis [57], IEC-Checker [24], eina d'analitzadors estàtics [58].	Alta. Ja integrat en IDEs industrials. Útil per a la majoria de programes.
Verificació Formal	Mètodes matemàtics per provar la correcció respecte a especificacions.	Garantir la correcció lògica i la seguretat del programa.	Model Checking (PLCverif [9], K-ST [29], nuXmv [28], Uppaal [77]), Màquines B [33].	Mitjana a Alta. Ideal per a sistemes crítics on l'error és inacceptable. Pot ser complex i costós.
Validació per Minería d'Especificacions	Extraure especificacions implícites del codi per a la seva validació posterior.	Fer la verificació més accessible i contextualitzada.	Approaches basades en mineria d'especificacions [3], pipelines iteratius com LLM4PLC [22].	Baixa a Mitjana. Àrea de recerca avançada, però amb un gran potencial per a futurs sistemes intel·ligents.

En conclusió, el paisatge de la verificació per a programes IEC 61131-3 ofereix eines per a tots els nivells de criticitat. L'anàlisi estàtic proporciona una barreja baixa de barreja ideal per a la majoria dels desenvolupadors, mentre que la verificació formal ofereix garanties matemàtiques de correcció per a aplicacions de seguretat crítica. Un sistema integral hauria de oferir múltiples nivells de verificació, permetent als usuaris escalar la profunditat de la seva validació en funció del risc associat al seu sistema d'automatització.

Integració en Entorns Industrials: Desafiaments i Oportunitats

El disseny d'un sistema per a la generació i verificació de codi sCL/ST ha de tenir clar el seu context d'aplicació final: un entorn industrial real o un entorn acadèmic independent. Aquesta decisió té implicacions profundes en l'arquitectura, les funcionalitats i els criteris de success del projecte. Donat que l'estàndard IEC 61131-3 és un instrument internacional per a la programació de controladors lògics programables [10] [12], la direcció més pragmàtica i amb major impacte seria centrar-se en el context industrial. Aquest entorn implica una necessitat imperiosa d'integració fluida amb les plataformes de desenvolupament i els equips de treball existents en l'indústria.

Les plataformes de desenvolupament integrat (IDEs) com CODESYS i TwinCAT són mencionades repetidament com a ecosistemes dominants en el món de la programació PLC [2] [15] [41] [57]. Un sistema dissenyat per a l'entorn industrial ha de buscar una integració seamless amb aquestes eines. Això significa que les seves funcionalitats

podrien prendre la forma d'un complement o extensió per a aquests IDEs, en comptes d'intentar crear una eina autònoma que compitiu amb elles. CODESYS, per exemple, ja ofereix eines sofisticades com CODESYS Static Analysis per a l'anàlisi de codi [54] [55], i TwinCAT també inclou mecanismes per a la validació i configuració [57]. Un nou sistema hauria de mirar d'ampliar, en comptes de substituir, aquestes capacitats existents. Aquesta integració permetria als desenvolupadors accedir a les noves funcionalitats de generació i verificació directament dins de l'entorn familiar on ja desenvolupen i depuren el seu codi.

Un dels reptes més significatius en l'entorn industrial és la gestió de les implementacions específiques dels proveïdors. Encara que l'estàndard IEC 61131-3 defineix una sintaxi i semàntica bàsiques, els fabricants hi afegeixen extensions, funcions i comportaments únics [2]. Per exemple, el manual de referència per a la programació SCL per a Siemens S7-300/S7-400 descriu extensions específiques a la norma [30], i el projecte AutoPLC ha de tenir en compte les diferències entre la versió oberta de ST de CODESYS i la dialecta SCL de Siemens [15]. Un sistema que sigui "aware of vendor" podria ser una diferenciació clau. Permetria als usuaris seleccionar el seu proveïdor i equip, i el sistema generaria codi que no només segueixi l'estàndard general, sinó que també fa servir de manera optimitzada les funcionalitats especialitzades del seu hardware i software. Aquesta capacitat és essencial per a ser acceptat i útil en un entorn industrial real.

Altres consideracions pràctiques inclouen la compatibilitat amb formats de fitxers estàndard, com el `.st` per a fitxers de SCL [81], i la gestió de biblioteques de funcions (FBs), que són una part fonamental de la programació modular en PLCs [81]. L'objectiu final d'un sistema industrial és simplificar i accelerar el cicle de vida del software del controlador, no crear una nova frontera [23]. Per tant, el sistema hauria de ser eficient, fiable i fàcil d'integrar en els processos de desenvolupament existents. Els IDEs com CODESYS i TwinCAT ja fan servir l'estàndard IEC 61131-3 per a la seva comprovació de sintaxi [41], i un nou sistema podria afegir capes de verificació més profounds sobre aquesta base.

Per contrast, un sistema dissenyat per a un entorn acadèmic o independent té objectius i restriccions diferents. En aquest context, la prioritat pot ser la validació teòrica de noves tècniques, la creació de noves semàntiques formals o la benchmarking de diferents models de LLM [1]. Hi ha menys pressió per adaptar-se a les limitacions i especificitats del mercat comercial, i més llibertat per experimentar amb arquitectures innovadores. Per exemple, un projecte acadèmic podria centrar-se exclusivament en la implementació d'un nou algorisme de model checking per a un subconjunt del llenguatge ST, com ho ha fet amb la llengua Instruction List (IL) [66]. Aquest enfocament permet anar més enllà del

que està disponible en les eines comercials i contribuir al coneixement fonamental del camp.

En definitiva, la tria entre un enfocament industrial o acadèmic defineix la trajectòria del projecte. Un enfocament industrial, centrat en la integració amb plataformes com CODESYS i TwinCAT i la gestió de l'awareness de proveïdor, té un potencial d'impacte directe i massiu en la productivitat i la fiabilitat de la indústria. Un enfocament acadèmic, per la seva banda, pot portar a innovacions teòriques que poden alimentar el desenvolupament de les eines industrials futures. Considerant l'objectiu declarat de generar i/o comprovar programes per a automatització industrial, una estratègia híbrida o un enfocament predominantement industrial sembla ser el més coherent i valuós.

Arquitectura Proposta i Fluxos de Treball Iteratius

La creació d'un sistema integrat per a la generació i verificació de programes sCL/ST conforme a l'estàndard IEC 61131-3 requereix una arquitectura que pugui gestionar de manera eficient tant la creativitat de la generació automàtica com la rigurositat de la verificació. L'anàlisi de les fonts i les tendències emergents suggerix que la direcció més potent és l'elaboració d'un flux de treball tancat en si mateix, on la generació i la verificació col·laboren en un bucle iteratiu per a refinar progressivament el codi resultant. Aquesta arquitectura dual no només millora la productivitat, sinó que també assegura una qualitat de codi superior, un requisit indispensable per a la fiabilitat en sistemes d'automatització industrial [29].

Una possible arquitectura per a aquest sistema podria estar dividida en tres capes principals: una capa d'interfície d'usuari (UI), una capa de processament central i una capa de connectors o adaptadors. La capa d'interfície d'usuari podria prendre la forma d'un editor de text intel·ligent o, de manera més integrada, com un complement per a IDEs existents com CODESYS o TwinCAT. Aquesta interfície permetria a l'usuari introduir requisits (potser en llenguatge natural o mitjançant diagrames visuals), visualitzar el codi generat i rebre retroalimentació immediata de les etapes de verificació. La capa de processament central seria el nucli del sistema, dividit en dos mòduls principals: un generador i un verificador. El generador podria ser un sistema basat en Models Largs de Llenguatge (LLM), inspirat en frameworks com Agents4PLC [37] o LLM4PLC [38], que transforma els requisits en codi ST/SCL. Aquest mòdul hauria de ser conscient del proveïdor per adaptar-se a les extensions de cada plataforma [2] [15]. El mòdul de verificació, per la seva banda, seria una eina d'anàlisi estàtic robusta, similar a la

implementada en CODESYS Static Analysis [54], capaç de detectar problemes com variables no inicialitzades, codi mort o violacions de convencions de codificació [24] [58]. A més, aquest mòdul podria incloure opcionalment una interfície a eines de verificació formal com PLCverif [9] o K-ST [29] per a una validació més profunda.

El flux de treball tancat en si mateix seria la característica més distintiva d'aquest sistema. En lloc de tractar la generació i la verificació com a passes separades, el sistema les fusionaria en un únic cicle iteratiu. El procés podria ser el següent: 1. **Entrada de Requisits:** L'usuari proporciona un conjunt de requisits per al nou control (p. ex., "El motor s'ha d'encendre si el sensor A és verd i el sensor B no és actiu"). 2. **Generació Inicial:** El sistema genera una primera versió del codi ST/SCL corresponent. 3. **Verificació Estàtica Ràpida:** Aquest codi generat és immediatament analitzat per la capa d'anàlisi estàtic. Si s'identifiquen errors sintàctics o problemes de qualitat, el sistema genera un missatge de retroalimentació específica i torna a generar el codi. 4. **Refinament Iteratiu:** El sistema pot incorporar un mecanisme de feedback tancat, tal com es demostra en l'enfocament de LLM4PLC [38]. Si l'anàlisi estàtic detecta un problema (p. ex., una variable no inicialitzada), aquesta informació es torna a alimentar al generador LLM per a una correcció iterativa. El projecte Agents4PLC ja demostra que aquest tipus de cicle tancat és viabilitzable i efectiu [37] [68]. 5. **Verificació Formal (Opcional):** Per a codi crític, el sistema podria oferir una opció per a una verificació formal. El codi validat estàticament es traduiria a un format adequat per a una eina de model checking (p. ex., Promela per a SPIN via la semàntica de poST [67]) i es comprovarien propietats lògiques específiques. 6. **Entrega del Resultat:** Finalment, el sistema retorna al usuari un codi que ha estat validat estàticament i, opcionalment, formalment, preparat per a la seva integració en el projecte.

Desenvolupar aquest sistema de manera incremental seria una estratègia prudent. Una possible cronologia seria:

- **Fase 1 (Verificació):** Començar per desenvolupar una eina d'anàlisi estàtic d'alta qualitat. Aquesta podria ser una API autònoma o una extensió per a IDEs, centrada en la detecció de problemes complexos [58].
- **Fase 2 (Generació):** Implementar una primera versió de generació basada en LLM, potser com una API independent, que segueixi la filosofia de frameworks com Spec2Control [69].
- **Fase 3 (Integració):** Crear el pipeline tancat que connecti la Fase 1 i la Fase 2, permetent un flux de treball iteratiu i reflexiu.

- **Fase 4 (Extensió):** Incorporar opcionalment una capa de verificació formal mitjançant integracions amb eines com PLCverif [9] o K-ST [29], dirigida a casos d'ús de fiabilitat extrema.

Finalment, l'objectiu d'aquest sistema no ha de ser simplement generar codi ràpidament, sinó generar codi de qualitat. El valor resideix en la fiabilitat, mantenibilitat i eficiència del resultat final. El sistema hauria de promoure activament bones pràctiques de programació [55], analitzar metriques de qualitat [79] i assegurar-se que el codi generat compleixi amb les especificacions funcionals originals. Centrar-se en aquests aspectes de qualitat, en comptes de només en la velocitat de generació, és el que convertiria aquest sistema en una eina de gran valor per a la comunitat d'enginyeria de control industrial.

---

en

1. Benchmarking and validation of prompting techniques for AI ... https://www.sciencedirect.com/science/article/pii/S2666827025001872
2. AutoPLC: Generating Vendor-Aware Structured Text for ... - arXiv https://arxiv.org/html/2412.02410v2
3. A User-Friendly Verification Approach for IEC 61131-3 PLC Programs https://www.mdpi.com/2079-9292/9/4/572
4. Automated test generation for IEC 61131-3 ST programs via ... https://www.sciencedirect.com/science/article/pii/S0167642321000010
5. Enhancing the Resilience of IEC 61131-3 Software with Online ... https://ieeexplore.ieee.org/iel8/8856/4358066/10892232.pdf
6. [PDF] LLM4PLC: Harnessing Large Language Models for Verifiable ... - arXiv https://arxiv.org/pdf/2401.05443
7. Modeling, Verification, and Implementation of PLC Program using ... https://dl.acm.org/doi/pdf/10.5555/1357910.1357994
8. Towards the automatic verification of PLC programs written in ... https://ieeexplore.ieee.org/document/884359/
9. (PDF) PLCverif: A tool to verify PLC programs based on model ... https://www.researchgate.net/publication/298274708_PLCverif_A_tool_to_verify_PLC_programs_based_on_model_checking_techniques

10. [PDF] Standards Compliance according to IEC 61131-3 - Support https://support.industry.siemens.com/cs/attachments/50204938/IEC_61131_compliance_e.pdf
11. [PDF] INTERNATIONAL STANDARD IEC 61131-3 https://d1.amobbs.com/bbs_upload782111/files_31/ourdev_569653.pdf
12. [PDF] Standards compliance according to IEC 61131-3 (3rd Edition) https://cache.industry.siemens.com/dl/files/748/109476748/att_845621/v1/IEC_61131_compliance_en_US.pdf
13. Control Server - an overview | ScienceDirect Topics https://www.sciencedirect.com/topics/engineering/control-server
14. Is Iec 61508 6 2000 | PDF | Systems Science | Safety - Scribd https://www.scribd.com/document/707091929/is-iec-61508-6-2000
15. [PDF] AutoPLC: Generating Vendor-Aware Structured Text for ... - arXiv https://arxiv.org/pdf/2412.02410
16. PLC orchestration automation to enhance human–machine ... https://www.sciencedirect.com/science/article/pii/S0278612523001474
17. Jim Hull - Freelance | LinkedIn https://www.linkedin.com/in/jim-hull-0868a94
18. Brandon Henley - OPHARDT hygiene - LinkedIn https://ca.linkedin.com/in/brandon-henley-a4aa154a
19. PLC 30 PDF | PDF | Programmable Logic Controller | Nuclear Reactor https://www.scribd.com/document/603910121/PLC-30-pdf
20. Jonny Wilson - Senior Staff Controls Engineer @ Rivian | LinkedIn https://www.linkedin.com/in/jonnylwilson
21. Devin Wine - Instrumentation & Controls Engineer - LinkedIn https://www.linkedin.com/in/devin-wine-12a03339
22. LLM4PLC: Harnessing Large Language Models for Verifiable ... - arXiv https://arxiv.org/html/2401.05443v1
23. [PDF] — PLC Automation PLCs, Control Panels, Engineering Suite AC500 ... https://search.abb.com/library/Download.aspx?DocumentID=3ADR020077C0204&LanguageCode=en&DocumentPartId=&Action=Launch
24. How to make troubleshooting smarter with automation - LinkedIn https://www.linkedin.com/posts/brandon-smith-b0b84844_%F0%9D%97%AA%F0%9D%97%B2-%F0%9D%97%BB%F0%9D%97%B2%F0%9D%97%B2%F0%9D%97%B1-%F0%9D%97%B9%F0%9D%97%AE%F0%9D%97%B1%F0%9D%97%B1%F0%9D%97%B2%F0%9D%97%BF-%F0%9D%97%B9%F0%9D%97%BC%F0%9D%97%B4%F0%9D%97%B6%F0%9D%9

7%B0-%F0%9D%97%AE%F0%9D%97%BB%F0%9D%97%B1-activity-7322650851829809152-LwwD

25. 3 Software With Online Reconfigurations for Fault Handling https://ieeexplore.ieee.org/iel8/8856/10839176/10892232.pdf
26. Understanding the SCL Programming Interface in TIA Portal - LinkedIn https://www.linkedin.com/posts/solisplc_understanding-the-scl-programming-interface-activity-7399462245786865664-pmQP
27. IEC 61131-3: The Standard Behind PLC and DCS Programming https://www.linkedin.com/posts/reza-abbasinejad-a784112b_industrialautomation-plc-dcs-activity-7367808710129192961-P4QG
28. [PDF] Agents4PLC: Automating Closed-loop PLC Code Generation ... - arXiv https://arxiv.org/pdf/2410.14209
29. KST: Executable Formal Semantics of IEC 61131-3 Structured Text ... https://ieeexplore.ieee.org/iel7/6287639/8600701/08620198.pdf
30. [PDF] Structured Control Language (SCL) for S7-300/S7-400 Programming https://cache.industry.siemens.com/dl/files/188/1137188/att_27471/v1/SCLV4_e.pdf
31. [PDF] A Formal Executable Semantics of PLC Structured Text Language https://arxiv.org/pdf/2202.04076
32. Understanding Structured Control Language (SCL) Elements https://www.linkedin.com/posts/solisplc_understanding-structured-control-language-activity-7391852209766047744-Sra7
33. [PDF] Formal Verification of PLC Programs Using the B Method ... https://www.semanticscholar.org/paper/Formal-Verification-of-PLC-Programs-Using-the-B-Barbosa-D%C3%A9harbe/a3885deb9ecb5f935f46ec9f3d385185613f6a6e
34. Programming embedded devices in IEC 61131-languages with ... https://www.researchgate.net/publication/272773580_Programming_embedded_devices_in_IEC_61131-languages_with_industrial_PLC_tools_using_PLCopen_XML
35. Formal verification of function blocks applied to IEC 61131-3 https://www.sciencedirect.com/science/article/pii/S0167642315002981
36. [PDF] Overview of the IEC 61131 Standard - ABB https://library.e.abb.com/public/81478a314e1386d1c1257b1a005b0fc0/2101127.pdf
37. Agents4PLC: Automating Closed-loop PLC Code Generation ... - arXiv https://arxiv.org/html/2410.14209v1
38. LLM4PLC: Harnessing Large Language Models for Verifiable ... https://dl.acm.org/doi/10.1145/3639477.3639743
39. IEC 61131-3:2025标准最新版本（4th）出炉，IL语言已成弃子！ https://cloud.tencent.com/developer/article/2622241

40. (PDF) KST: Executable Formal Semantics of IEC 61131-3 Structured ... https://www.researchgate.net/publication/330540744_KST_Executable_Formal_Semantics_of_IEC_61131-3_Structured_Text_for_Verification
41. Agents4PLC: Automating Closed-loop PLC Code Generation and ... https://arxiv.org/html/2410.14209v2
42. Translation of continuous function charts to imperative synchronous ... https://dl.acm.org/doi/10.1145/3487212.3487338
43. Automated Control Logic Test Case Generation using ... - arXiv.org https://arxiv.org/html/2405.01874v1
44. Automatic generation of a PLC controller based on a control system ... https://www.researchgate.net/publication/351571779_Automatic_generation_of_a_PLC_controller_based_on_a_control_system-identified_model
45. Five programming languages that describe the PLC IEC 61131-3 ... https://en.eeworld.com.cn/news/qrs/eic649388.html
46. A Model-Based Approach to Automated Validation and Generation ... https://www.mdpi.com/2076-3417/12/15/7506
47. (PDF) PLC orchestration automation to enhance human-machine ... https://www.researchgate.net/publication/374025566_PLC_orchestration_automation_to_enhance_human-machine_integration_in_adaptive_manufacturing_systems
48. [PDF] Standards Compliance according to IEC 61131-3 - Support https://support.industry.siemens.com/cs/attachments/8790932/norm_tbl.pdf
49. A syntactic specification for the programming languages of the IEC ... https://www.academia.edu/25472082/A_syntactic_specification_for_the_programming_languages_of_the_IEC_61131_3_standard
50. IEC 61131-3 Standards Overview | PDF - Scribd https://www.scribd.com/document/482710971/IEC-61131-compliance-pdf
51. [PDF] TECHNICAL REPORT IEC TR 61131-8 https://d1.amobbs.com/bbs_upload782111/files_31/ourdev_569657.pdf
52. [PDF] Standards Compliance according to IEC 61131-3 - Support https://support.industry.siemens.com/cs/attachments/43208859/IEC_61131_compliance.pdf
53. Safety Verification of IEC 61131-3 Structured Text Programs https://pure.ecnu.edu.cn/en/publications/safety-verification-of-iec-61131-3-structured-text-programs

54. CODESYS Static Analysis https://content.helpme-codesys.com/en/CODESYS%20Static%20Analysis/_san_start_page.html
55. CODESYS Static Analysis Concept - Schneider Electric https://product-help.schneider-electric.com/Machine%20Expert/V2.2/en/CODESYS_Static_Analysis/CODESYS_Static_Analysis/modules/_san_static_analysis_concept.html
56. [PDF] CODESYS® for Users - Messe Frankfurt https://exhibitorsearch.messefrankfurt.com/images/original/userdata/bata/204697/5d5149a9018f1.pdf
57. TwinCAT IEC61131-3 Guide | PDF | Real Time Computing | Data Type https://www.scribd.com/document/185069646/127955753-Beckhoff-IEC61131-3
58. Static Code Analysis of IEC 61131-3 Programs - Semantic Scholar https://www.semanticscholar.org/paper/Static-Code-Analysis-of-IEC-61131-3-Programs%3A-Tool-Pr%C3%A4hofer-Angerer/4c43f5b85040cd1fa10a3e3258173c758e324748
59. PLC Controls with Structured Text (ST), V3: IEC 61131-3 and best ... https://www.researchgate.net/publication/344224775_PLC_Controls_with_Structured_Text_ST_V3_IEC_61131-3_and_best_practice_ST-programming
60. Part 3: Automatic Code Generation for PLCs - MathWorks https://www.mathworks.com/videos/automatic-code-generation-for-plcs-1697470799317.html
61. PLC Code Generation in the Development Process - MathWorks https://www.mathworks.com/help/plccoder/gs/plc-code-generation-in-the-development-process.html
62. Design patterns for model-based automation software design and ... https://www.researchgate.net/publication/271606886_Design_patterns_for_model-based_automation_software_design_and_implementation
63. IEC 61131-3 model for model-driven development - Semantic Scholar https://www.semanticscholar.org/paper/IEC-61131-3-model-for-model-driven-development-Wenger-Zoitl/740061466f60d8c886ab889df44a022d0606cb19
64. Poster: Model-based design of time-triggered real-time embedded ... https://www.researchgate.net/publication/283824150_Poster_Model-based_design_of_time-triggered_real-time_embedded_systems_for_digital_manufacturing
65. Model Checking Programs in Process-Oriented IEC 61131-3 ... https://dl.acm.org/doi/abs/10.3103/S0146411624700433
66. [PDF] A formal semantics of PLC programs in Coq - HAL-Inria https://inria.hal.science/inria-00601906/document

67. Model Checking Process-Oriented Iec 61131-3 Structured Text ... https://www.academia.edu/110950142/Model_Checking_Process_Oriented_Iec_61131_3_Structured_Text_Programs
68. (PDF) Agents4PLC: Automating Closed-loop PLC Code Generation ... https://www.researchgate.net/publication/385091923_Agents4PLC_Automating_Closed-loop_PLC_Code_Generation_and_Verification_in_Industrial_Control_Systems_using_LLM-based_Agents
69. Spec2Control: Automating PLC/DCS Control-Logic Engineering ... https://arxiv.org/html/2510.04519v1
70. Verification case definition form. - ResearchGate https://www.researchgate.net/figure/erification-case-definition-form_fig2_298274708
71. Training LLMs for Generating IEC 61131-3 Structured Text ... - arXiv https://arxiv.org/html/2410.22159v1
72. Advanced techniques for customizing IEC 61131 code generation ... https://www.mathworks.com/videos/advanced-techniques-for-customizing-iec-61131-code-generation-for-plcs-1607094156796.html
73. KST: Executable Formal Semantics of IEC 61131-3 Structured Text ... https://www.semanticscholar.org/paper/KST%3A-Executable-Formal-Semantics-of-IEC-61131-3-for-Huang-Bu/2f83b16533aab6d56760523f270debae5e07552e
74. A Formal Executable Semantics of PLC Structured Text Language https://www.researchgate.net/publication/358490876_K-ST_A_Formal_Executable_Semantics_of_PLC_Structured_Text_Language
75. Safety Verification of IEC 61131-3 Structured Text Programs https://www.semanticscholar.org/paper/Safety-Verification-of-IEC-61131-3-Structured-Text-Xiong-Bu/8135c0fad1e506259e3c9760a9e5668f71e0455b
76. A Survey of Static Formal Methods for Building Dependable ... https://www.researchgate.net/publication/353863223_A_Survey_of_Static_Formal_Methods_for_Building_Dependable_Industrial_Automation_Systems
77. A Systematic Review on the Applications of Uppaal - MDPI https://www.mdpi.com/1424-8220/25/11/3484
78. A Survey on LLM-based Code Generation for Low-Resource and ... https://arxiv.org/html/2410.03981v3
79. Assessment of the PLC Code generated with the GEMMA ... https://www.sciencedirect.com/science/article/pii/S1877050922002770/pdf?md5=54e1505c1aa1eee1007dfe86b54cca12&pid=1-s2.0-S1877050922002770-main.pdf

80. Intelligent PLC Code Generation in HCPS 2.0: A Multi-dimensional ... https://dl.acm.org/doi/10.1145/3728725.3728757

81. Design and implementation of O-PAS user-defined function blocks https://link.springer.com/article/10.1186/s43067-024-00183-9

82. Evolution of software in automated production systems: Challenges ... https://www.sciencedirect.com/science/article/pii/S0164121215001818